2010

COMPOSITE KERNEL FEATURE ANALYSIS FOR CANCER CLASSIFICATION

Sindhu Myla
Virginia Commonwealth University

COMPOSITE KERNEL FEATURE ANALYSIS FOR CANCER CLASSIFICATION

A Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering at Virginia Commonwealth University.

by
SINDHU MYLA
Bachelor of Technology, Jawaharlal Nehru Technological University, India, 2008
Director: YUICHI MOTAI
ASSISTANT PROFESSOR, ELECTRICAL ENGINEERING
Virginia Commonwealth University
Richmond, Virginia
May 2010

Acknowledgements

This thesis could not have been completed without the help and guidance of number of people. First of all I would thank God almighty. I would next like to thank my Husband G. Subhash Reddy for his immense support though out my Education. I would like to thank Dr. Yuich Motai, my thesis advisor, for the help, support and guidance he provided me during the research. I would also like to thank Dr. Alen Docef and Dr. Aimee Ellington for their insights. I also would like to thank Dr. Suprio Bandhopadhay, Dr. Ross Anderson, and Dr. Laura McLay for their wonderful courses. Last but not the least I would like to thank Dr. Rosalyn Hobson and Dr. Ashok Iyer for their wonderful support during my stay at VCU.

It was a pleasure to have studied together with a great group of students who made my stay at the School of Engineering one of the most memorable events of my life.

Table of Contents

List of Tables

List of Figures

CHAPTER 1 INTRODUCTION

1.1 Introduction

Computed tomographic (CT) colonography, or virtual colonoscopy, is a promising technique for screening colorectal cancers by use of CT scans of the colon [1]. Current CT technology allows a single image set of the colon to be acquired in 10-20 seconds, which translates into an easier, more comfortable examination than is available with other screening tests. For CT colonography to be a clinically practical means of screening for colon cancers, the technique must be feasible for interpreting a large number of images in a time-effective fashion, and it must facilitate the detection of polyps—a precursor of colorectal cancers—with high accuracy. Currently, however, interpretation of an entire CT colonography examination is time-consuming, and the reader performance for polyp detection varies substantially [2, 3].To overcome these difficulties while providing a high detection performance of polyps, researchers are developing computer-aided detection (CAD) schemes that automatically detect suspicious lesions in CT colonography images [4]. CAD for CT colonography provides the locations of the suspicious polyps to radiologists, which offers a second opinion that has the potential to improve radiologists' detection performance.

1.2 Aim of the study

Polyps appear as bulbous, caplike structures that adhere to the colonic wall and protrude to the lumen, whereas folds appear as elongated, ridgelike structures, and the colonic wall appears as a large, nearly flat, cuplike structure. Therefore, most CAD schemes employ a model-based approach for the detection of polyp candidates, in which shape analysis

methods that differentiate among these distinct types of shapes plays an essential role [5-9]. Nevertheless, there are many naturally occurring normal colonic structures that occasionally imitate such shapes, and therefore the resulting polyp candidates typically include many false positives. The reduction of such false positives is often performed by first extracting a set of image features from segmented polyp regions, followed by application of a statistical classifier to the feature space for discrimination of false positives from actual polyps [8, 10-13]. Such CAD schemes tend to show a high sensitivity in the detection of polyps; however, they tend to suffer from a much large number of false positives than that of human readers [4].

The overall goal of this study is to achieve a high performance in the detection of polyps on CT colonographic images by effectively incorporating an appearance-based object recognition approaches into a model-based CAD scheme. The specific contribution of our studies is to develop a fast kernel feature analysis that, in combination with a shape-based polyp detection method, can efficiently differentiate polyps from false positives and thus improve the detection performance of polyps.

1.3 Key Idea

The key idea behind the proposed algorithm is to reconstruct a feature space by use of a feature mapping that maps the original, raw feature space into a higher dimensional feature space. Such a high-dimensional feature space is expected to have a greater classification power than that of the original feature space, as suggested by the Vapnik-Chervonenkis theory [14]. Also we developed a novel method of selecting kernel functions that are appropriate for the given data set and then use their linear combinationi in the construction of Kernel Gram matrix which can then used for efficient reconstruction of feature space. We evaluated our fast kernel feature analysis on texture-based features that were extracted from the polyp candidates generated by our shape-based CAD scheme. The main contribution of this work lies in providing a Composite kernel Matrix that involves appearance-based approach to improve kernel feature analysis for the classification of texture-based features.

CHAPTER 2 APPEARANCE-BASED RECOGNITION FOR CAD IN CT COLONOGRAPHY

2.1 Introduction

The development of conventional CAD schemes is usually based on model-based recognition, where the design of the detection methods is based on the use of explicit models of the target lesions. For example, the fully automated CAD scheme that was developed in Ref. [5] represents model-based recognition. The CAD scheme extracts the colonic wall based on an anatomical model of the colon, and compares the local shape characteristics of the extracted region with that of a polypoid shape model. The polypoid shape model was based upon two volumetric rotation-invariant shape features, the shape index (SI) and the curvedness (CV). The SI feature characterizes the topological 3-D shape of a local iso-intensity surface in the vicinity of a voxel, and the CV feature characterizes the flatness of that shape. They can be calculated from the principal curvatures κ_1 and κ_2 of the local 3-D iso-surface at voxel p as follows:

$$SI(p) \equiv -\frac{2}{\pi}\arctan\frac{\kappa_1(p)+\kappa_2(p)}{\kappa_1(p)-\kappa_2(p)},$$

$$CV(p) \equiv \sqrt{\frac{1}{2}(\kappa_1(p)^2+\kappa_2(p)^2)}.$$

Because of the high detection sensitivity of the CAD scheme, it also includes a Bayesian neural network model to reduce false-positive (FP) detections, where the input features have been modeled to identify typical false positives. For example, a 3-D gradient concentration feature is used to identify folds based on the concentration of the gradient vectors in the vicinity of an operating point. Although model-based CAD schemes can

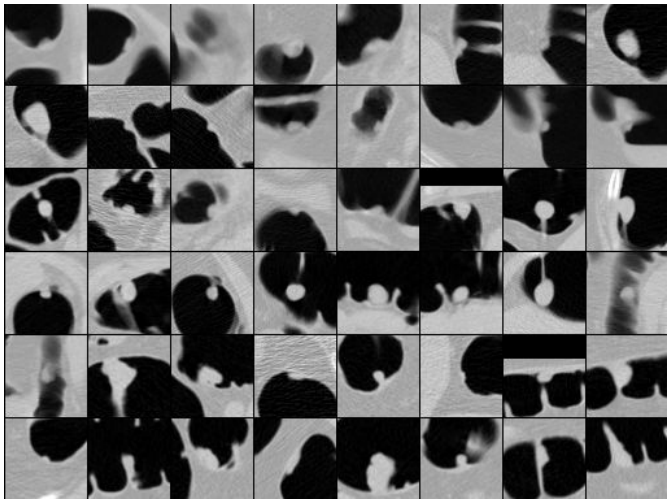

Fig. 2.1. Sample images of polyps on CT colonographic images.

provide excellent detection performance, such schemes have several fundamental limitations. The model-based paradigm requires explicit mathematical modeling of the problem; however, it is often not obvious how to develop such a model with sufficient detail for practical applications in a clinical setting. This makes the development of model-based systems a demanding and time- consuming task that requires a considerable amount of human labor. For example, Fig. 1 shows images of 48 polyps in our database of CT colonographic images. These images demonstrate that the shape of polyps varies substantially across different polyps; thus the characterization and detection of polyps based only on the shape information is limited. Furthermore, a specific model that is developed for a specific application tends to generalize poorly to other problems. Finally, despite the progress made in CAD methodology during the past years, the CAD schemes that are currently used in a clinical setting tend to generate a large number of FP detections that irritate experienced readers.

Appearance-based recognition is another widely used computer vision methodology, although, currently, it is rarely applied for medical imaging applications. We are developing an appearance-based method that complements the performance of a model-based CAD scheme in the detection of polyps. In this paper, we propose using an appearance-based method for texture-based feature analysis of the polyp candidates, obtained by a shape-based method, for discrimination of false positives from true polyps. Unlike the model-based approaches, the texture-based feature analysis does not specify

the solid target model; instead, the texture features are simply handled as instances in the analysis process.

2.2 Kernel Basics

For efficient feature analysis, extraction of the salient features of polyps is essential because of the size and the 3-D nature of the polyp datasets. Moreover, the distribution of the image features of polyps is expected to be non-linear. Therefore, we employ Kernel Principal Component Analysis (Kernel PCA) [15, 16, 19, 20, 21], which is well known as a superior data compression method, and its extension to a non-linear feature space, kernel feature space, in this paper.

Fig. 2 illustrates the concept of feature dissimilarity and of classifier selection by use of the kernel feature space. Here, Fig. 2(a) shows a set of linearly inseparable two-class features in the input space, in which features belonging to a class is labeled by green and those belonging to the other class is labeled by red, and Fig. 2(b) shows a set of linearly separable features using a hyper plane. The features in Fig. 2(a) are transferable to the feature set in Fig. 2(b) by converting the input space into an appropriate higher-dimensional feature space using an operator, so that the features in the two classes can be linearly separable.

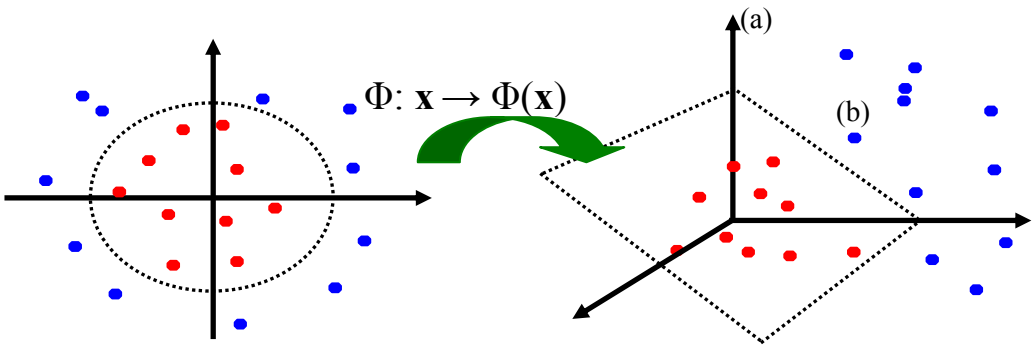

Figure 2.2. Feature distribution illustration with color-encoded class labels. The red and blue colors indicate class labels. These binary classes are linearly separable by mapping them into a higher-dimensional (kernel) feature space.

The problem is how to select such an ideal operator. A nonlinear, positive-definite kernel $k : R^d \times R^d \to R$ of an integral operator, e.g., $k(x, y) = \exp\left\{-\|x - y\|^2\right\}$, computes the inner product of the transformed vectors $\langle \Phi(x), \Phi(y) \rangle$, where $\Phi : R^d \to H$ denotes a nonlinear embedding (induced by *k*) into a possibly infinite dimensional Hilbert space *H*. Given n sample points in the domain $X_n = \{x_i \in R^d \mid i = 1, ..., n\}$, the image $Y_n = \{\Phi(x_i) \mid i = 1, ..., n\}$ of X_n spans a linear subspace of at most $(n-1)$ dimensions. Heuristically, the dominant linear correlations in the distribution of the image Y_n may elucidate important nonlinear dependencies in the original data sample X_n. This is advantageous because it permits making PCA non-linear without complicating the original PCA algorithm. The kernel function *k* is traditionally chosen in the form of a Gaussian function such as Radial Basis Functions (RBF): $k(x, y) = \exp\left\{-\|x - y\|^2 / 2\sigma^2\right\}$. This is illustrated in the following figure 3.

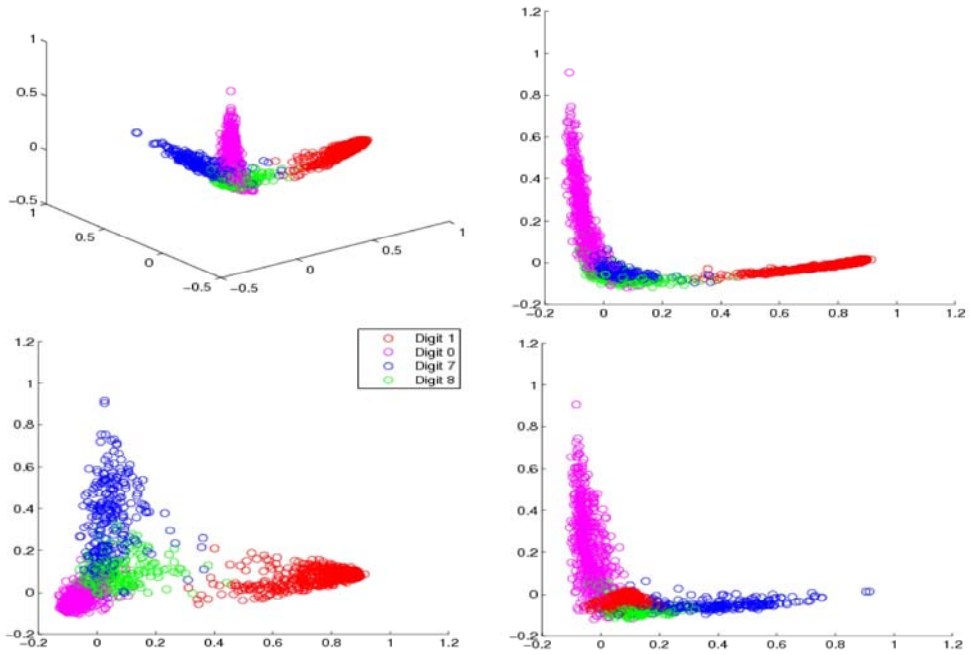

Figure 2.3. Feature distribution the simulation data in a Hilbert space.
Red, green, pink, and blue colors indicate different classes to which the feature points belong. These four-class features are separable after mapping them by RBF kernel function with σ = 4 into a higher-dimensional feature space.

As visually demonstrated in Fig. 2.3, when feature points in the input space are mapped, via the RBF kernel function, to higher-dimensional space corresponding to an inner product in an expanded feature space, features belonging to different classes can be well clustered, and thus the kernel space can be efficient in discriminating one class, e.g., false positives, from the other class, e.g., true polyps.

CHAPTER 3 BACKGROUND WORK

3.1 Kernel Feature Extraction Introduction:

Our new kernel feature analysis for CT colonographic images is based on the following two methods: Kernel Principal Feature Analysis and Sparse Kernel Feature Analysis. This section provides a brief review of these existing methods. In the next section, we will extend these existing methods and propose a new method called Accelerated Kernel Feature Analysis.

3.2 Kernel Principal Feature Analysis

Kernel Principal Component Analysis (KPCA) uses a Mercer kernel [15] to perform a linear principal component analysis of this transformed image. Without loss of generality, we assume that the image of the data has been centered so that its scatter matrix in H is given by $S = \sum_{i=1}^{n} \Phi(x_i)\Phi(x_i)^T$. Eigenvalues λ_j and eigenvectors e_j are obtained by solving

$$\lambda_j e_j = S e_j = \sum_{i=1}^{n} \Phi(x_i)\Phi(x_i)^T e_j = \sum_{i=1}^{n} \left\langle e_j, \Phi(x_i) \right\rangle \Phi(x_i). \tag{1}$$

for $j = 1,...,n$. Since Φ is not known, (1) must be solved indirectly. Letting $a_{ji} = \frac{1}{\lambda_j}\left\langle e_j, \Phi(x_i) \right\rangle$ gives

$$e_j = \sum_{i=1}^{n} a_{ji}\Phi(x_i). \tag{2}$$

Multiplying by $\Phi(x_q)^T$ on the left, for $q = 1,\cdots,n$, yields

$$\lambda_j \left\langle \Phi(x_q), e_j \right\rangle = \sum_{i=1}^{n} \left\langle e_j, \Phi(x_i) \right\rangle \left\langle \Phi(x_q), \Phi(x_i) \right\rangle. \tag{3}$$

Substitution of (2) into (3) produces

$$\lambda_j \left\langle \Phi(x_q), \sum_{i=1}^{n} a_{ji}\Phi(x_i) \right\rangle = \sum_{i=1}^{n} \left(\sum_{k=1}^{n} \left\langle a_{jk}\Phi(x_k), \Phi(x_i) \right\rangle \left\langle \Phi(x_q), \Phi(x_i) \right\rangle \right). \tag{4}$$

which can be rewritten as, $\lambda_j K a_j = K^2 a_j$, where K is an $n \times n$ Gram matrix, with the element $k_{ij} = \left\langle \Phi(x_i), \Phi(x_j) \right\rangle$ and $a_j = \left[a_{j1} a_{j2} \cdots a_{jn} \right]^T$. The latter is a dual eigenvalue problem equivalent to the problem

$$\lambda_j a_j = K a_j. \tag{5}$$

Since $\left\| e_j \right\|^2 = \left\langle \sum_{i=1}^{n} a_{ji}\Phi(x_i), \sum_{i=1}^{n} a_{ji}\Phi(x_i) \right\rangle = \left\langle a_j, K a_j \right\rangle = \lambda_j \left\| a_j \right\|^2$, the normalization of each eigenvector ($\left\| e_j \right\| = 1$) requires $\left\| a_j \right\|^2 = 1/\lambda_j$.

In the following, for the purpose of illustration, we choose a Gaussian kernel, i.e.,

$$k_{ij} = \left\langle \Phi(x_i), \Phi(x_j) \right\rangle = \exp\left(-\frac{1}{2\sigma^2} \left\| x_i - x_j \right\|^2 \right). \tag{6}$$

If the image of X_n is not centered in the Hilbert space, we need to use the centered Gram Matrix deduced by Smola, Mangasarian, and Schölkopf [16]:

$$\hat{K} = K - KT - TK + TKT \ . \tag{7}$$

where *K* is the Gram Matrix of uncentered data, and

$$T = \begin{bmatrix} \frac{1}{n} & \cdots & \frac{1}{n} \\ \cdots & \cdots & \cdots \\ \frac{1}{n} & \cdots & \frac{1}{n} \end{bmatrix}_{n \times n}$$

Keeping the ℓ eigenvectors associated with the ℓ largest eigenvalues, we can reconstruct data in the mapped space: $\Phi_i^{'} = \sum_{j=1}^{\ell} \left\langle \Phi_i, e_j \right\rangle e_j = \sum_{j=1}^{\ell} \beta_{ji} e_j$, where $\beta_{ji} = \left\langle \Phi_i, \sum_{k=1}^{n} a_{jk} \Phi_k \right\rangle = \sum_{k=1}^{n} a_{jk} k_{ik}$. The reconstruction square error of each data $\Phi_i, i = 1, \cdots, n$, is $\mathrm{Err}_i = \left\| \Phi_i - \Phi_i^{'} \right\|^2 = k_{ii} - \sum_{j=1}^{\ell} \beta_{ji}^2$. The mean square error is $\mathrm{MErr} = \frac{1}{n} \sum_{i=1}^{n} \mathrm{Err}_i$. Using (5), $\beta_{ji} = \lambda_j a_{ji}$. Therefore, the mean square reconstruction error is $\mathrm{MErr} = \frac{1}{n} \sum_{i=1}^{n} (k_{ii} - \sum_{j=1}^{\ell} \lambda_j^2 a_{ji}^2)$. Since $\sum_{i=1}^{n} k_{ii} = \sum_{i=1}^{n} \lambda_i$ and $\sum_{i=1}^{n} a_{ji}^2 = \left\| a_j \right\|^2 = 1/\lambda_j$, $\mathrm{MErr} = \frac{1}{n} \sum_{i=\ell+1}^{n} \lambda_i$.

Kernel PCA can now be summarized as follows:

Step 1: Calculate the Gram matrix using (6), which contains the inner products between pairs of image vectors.

Step 2: Use (5) to get the coefficient vectors a_j for $j = 1, ..., n$.

Step 3: The projection of a test point $x \in R^d$ along the *j*-th eignevector is

$$\left\langle e_j, \Phi(x) \right\rangle = \sum_{i=1}^{n} a_{ji} \left\langle \Phi(x_i), \Phi(x) \right\rangle = \sum_{i=1}^{n} a_{ji} k(x, x_i).$$

The above implicitly contains an eigenvalue problem of rank n, so the computational complexity of Kernel PCA is $O(n^3)$. In addition, each resulting eigenvector is represented as a linear combination of *n* terms. Thus, all data contained in X_n must be retained, which is computationally cumbersome and unacceptable for incremental or on-line learning.

3.3 Sparse Kernel Feature Analysis

Sparse Kernel Feature Analysis (SKFA) [16] extracts the features one by one in order of decreasing projection variance. SKFA improved the computational costs of KPCA, associated with both time complexity and data retention requirements. In that sense, SKFA is more compact and time efficient method than KPCA. The particular advantage of SKFA is that the ℓ features only depend on ℓ elements of X_n, which is extremely useful for on-line learning. We let $v_i \in H$, for $i = 1, ..., \ell$ denote the features selected by SKFA. We intentionally avoid using e_i in order to distinguish these features from the eigenvectors obtained using KPCA. Again, we analyze the scatter matrix of the image data. Following (1), with e_j replaced by v_j, we obtain $v_j^T \lambda_j v_j = v_j^T (\sum_{i=1}^{n} \left\langle v_j, \Phi(x_i) \right\rangle \Phi(x_i))$, where v_j is the *j*-th feature with unit length. Thus, $\lambda_j = \sum_{i=1}^{n} \left\langle v_j, \Phi(x_i) \right\rangle^2$. Therefore, the first feature, corresponding to the maximum eigenvalue, is chosen as the direction with the maximum projected variance:

$$v_1 = \arg\max_{\|v\|^2=1} \frac{1}{n} \sum_{i=1}^{n} |\left\langle v, \Phi(x_i) \right\rangle|^2 . \tag{8}$$

The global solution of (8), v_1, needs to satisfy an l_2 normalization constraint, i.e., unit Euclidian length. Changing the l_2 constraint to an l_1 constraint leads to a vertex solution. The l_1 constraint assumed by SKFA is

$$V_{l_1} = \left\{ \sum_{j=1}^{n} a_j \Phi_j \mid \sum_{j=1}^{n} \mid a_j \mid \leq 1 \right\}. \tag{9}$$

The first feature selected by SKFA satisfies

$$v_1 = \arg\max_{v \in V_{l_1}} \frac{1}{n} \sum_{i=1}^{n} | \langle v, \Phi(x_i) \rangle |^2 .$$

Smola et al. showed that this feature corresponds to an element of the image Y_n, hence

$$v_1 = \arg\max_{\Phi(x_j) \in V_{l_1}} \frac{1}{n} \sum_{i=1}^{n} \left| \langle \Phi(x_j), \Phi(x_i) \rangle \right|^2 . \tag{10}$$

Subsequent features are obtained iteratively. Suppose $i-1$ features $\{v_t \in H \mid t = 1, ..., i-1\}$ have been found; then each image $\Phi_j = \Phi(x_j)$ is projected into the orthogonal subspace to obtain

$$\Phi_j^i = \Phi_j - \sum_{t=1}^{i-1} v_t \frac{\langle \Phi_j, v_t \rangle}{\|v_t\|^2} = \Phi_j - \sum_{t=1}^{i-1} \frac{a_{tj} v_t}{\|v_t\|^2}. \tag{11}$$

where $a_{tj} = \langle \Phi_j, v_t \rangle$. Then Φ_j^i is normalized by the l_1 constraint in (9). The projection variance of the normalized Φ_j^i with all $\Phi_k, k = 1, \cdots, n$ is then calculated. Finally, one identifies the maximum projection variance and selects the corresponding Φ_j^i as the *i*-th feature, v_i.

Based on the ℓ features extracted by SKFA, each training data in the mapped space can be reconstructed by

$$\Phi_i' = \sum_{j=1}^{\ell} \frac{\langle \Phi_i, v_j \rangle v_j}{\|v_j\|^2} = \sum_{j=1}^{\ell} \frac{a_{ji} v_j}{\|v_j\|^2}. \tag{12}$$

where a_{ji}, the projection of *i*-th training data on *j*-th feature, is stored after extracting the *j*-th feature. According to (11), the feature set $\{v_1, ..., v_\ell\}$ only depends upon the set of ℓ image vectors, $\{\Phi_{idx(i)}, i = 1, \cdots, \ell\}$, where $idx(i)$ denotes the subscript of the projected

image Φ_j^i that was selected when constructing v_i. Therefore, after training, we only need to retain the ℓ input vectors $\{x_{idx(i)} \in R^d \mid i = 1,...,\ell\}$, where ℓ is the number of features extracted.

In this way, SKFA extracts ℓ features, where one assumes $\ell << n$. As $O(in^2)$ operations are required to extract the *i*-th feature, the total computational cost for ℓ features is $O(\ell^2 n^2)$, which is an improvement over the $O(n^3)$ operations required by kernel PCA.

3.4 Accelerated Kernel Feature Analysis (AKFA)

To further improve the efficiency and accuracy of SKFA, we propose an Accelerated *Kernel Feature Analysis* (AKFA) that (*i*) saves computation time by iteratively updating the Gram Matrix, (*ii*) normalizes the images with the l_2 constraint before the l_1 constraint is applied, and (*iii*) optionally discards data that falls below a threshold magnitude δ during updates.

First (*i*), instead of extracting features directly from the original mapped space, AKFA extracts the *i*-th feature based on the *i*-th updated Gram matrix K^i, where each element has $k_{jk}^i = \left\langle \Phi_j^i, \Phi_k^i \right\rangle$. Since $\Phi_j^i = \Phi_j^{i-1} - v_{i-1} \left\langle \Phi_j^{i-1}, v_{i-1} \right\rangle$,

$$\begin{aligned} k_{jk}^i &= \left\langle \Phi_j^{i-1}, \Phi_k^{i-1} \right\rangle - \left\langle \Phi_j^{i-1}, v_{i-1} \right\rangle \left\langle \Phi_k^{i-1}, v_{i-1} \right\rangle \\ &= k_{jk}^{i-1} - \frac{\left\langle \Phi_j^{i-1}, \Phi_{idx(i-1)}^{i-1} \right\rangle \left\langle \Phi_k^{i-1}, \Phi_{idx(i-1)}^{i-1} \right\rangle}{\left\| \Phi_{idx(i-1)}^{i-1} \right\|^2} = k_{jk}^{i-1} - \frac{k_{j,idx(i-1)}^{i-1} k_{k,idx(i-1)}^{i-1}}{k_{idx(i-1),idx(i-1)}^{i-1}}. \end{aligned} \tag{13}$$

By updating the Gram Matrix, it is unnecessary to save the projection of each individual data on all previous features. The computational cost for extracting *i*-th feature becomes $O(n^2)$, instead of $O(in^2)$ as in SKFA.

The second (*ii*) improvement is to revise the l_1 constraint. As shown in (10), SKFA treats each individual sample data as a possible direction, and computes the projection variances with all data. Since SKFA includes its length in its projection variance calculation, it is biased to select vectors with larger magnitude. From the objective function (8), we know that we are actually looking for a direction with unit length. When we choose an image vector as a possible direction, we ignore the length and only consider its direction, which improves the accuracy of the features. Therefore, in our AKFA algorithm, we replace the l_1 constraint (9) by

$$V_{l_1}^i = \left\{ \sum_{j=1}^{n} a_j \frac{\Phi^i{}_j}{\left\| \Phi^i{}_j \right\|} \mid \sum_{j=1}^{n} \mid a_j \mid \leq 1 \right\}.$$

The *i*-th feature is extracted by

$$v_i = \arg\max_{v \in V_{l_1}^i} \frac{1}{n} \sum_{j=1}^{n} \left| \left\langle v, \Phi^i{}_j \right\rangle \right|^2.$$

Since v_i is extracted from Φ^i space, the solution is located on one of $\hat{\Phi}^i{}_j = \Phi^i{}_j / \left\| \Phi^i{}_j \right\|$ for $j = 1, \cdots, n$. Equation (10) reduces to

$$v_i = \arg\max_{\hat{\Phi}^i{}_j} \frac{1}{n} \sum_{t=1}^{n} \left| \left\langle \Phi^i_t, \hat{\Phi}^i{}_j \right\rangle \right|^2 = \arg\max_{\hat{\Phi}^i{}_j} \frac{1}{n k^i_{jj}} \sum_{t=1}^{n} k^i{}_{jt}{}^2. \tag{14}$$

Each $\hat{\Phi}^i{}_j$ satisfies the l_1 constraint, so the normalization step that appears in SKFA is not required. Let $\Phi^i_{idx(i)}$ denote the image vector corresponding to the *i*-th feature. Suppose we have selected $(i-1)$ features with $\mathbf{V}_{(\mathbf{i}-1)} = \Phi_{(\mathbf{i}-1)} \mathbf{C}_{(\mathbf{i}-1)}$, where $\mathbf{V}_{(\mathbf{i}-1)} = [v_1 v_2 \cdots v_{(i-1)}]$, $\Phi_{(\mathbf{i}-1)} = [\Phi_{idx(1)} \Phi_{idx(2)} \cdots \Phi_{idx(i-1)}]$, and $\mathbf{C}_{(\mathbf{i}-1)}$ is the coefficient matrix, which is upper-triangular. Then $\Phi^i_{idx(i)} = \Phi_{idx(i)} - \sum_{t=1}^{i-1} \left\langle \Phi_{idx(i)}, v_t \right\rangle v_t$. Let's study the second term:

$$\sum_{t=1}^{i-1} \left\langle \Phi_{idx(i)}, v_t \right\rangle v_t = \sum_{t=1}^{i-1} v_t v_t^T \Phi_{idx(i)} = \Phi_{(\mathbf{i}-1)} \mathbf{C}_{\mathbf{i}-1} \mathbf{C}^{\mathrm{T}}_{\mathbf{i}-1} \mathbf{K}_{\mathbf{idx(i)}}. \tag{15}$$

where $\mathbf{K}_{\mathbf{idx(i)}} = [k_{idx(i),idx(1)} k_{idx(i),idx(2)} \cdots k_{idx(i),idx(i-1)}]^T$. Therefore,

$$v_i = (\Phi_{idx(i)} - \Phi_{(\mathbf{i}-1)} \mathbf{C}_{\mathbf{i}-1} \mathbf{C}^{\mathrm{T}}_{\mathbf{i}-1} \mathbf{K}_{\mathbf{idx(i)}}) / \sqrt{k^i_{idx(i),idx(i)}}.$$

Let

$$\mathbf{C}_{i,i} = 1/\sqrt{k^i_{idx(i),idx(i)}}. \tag{16}$$

and

$$\mathbf{C}_{1:(i-1),i} = -\mathbf{C}_{i,i}\mathbf{C}_{(\mathbf{i}-1)}\mathbf{C}^{\mathrm{T}}_{(\mathbf{i}-1)}\mathbf{K}_{\mathbf{idx(i)}}. \tag{17}$$

Then, $\mathbf{V_i} = \Phi_\mathbf{i}\mathbf{C_i}$, where $\mathbf{V_i} = [\mathbf{V}_{(\mathbf{i}-1)}, v_i]$, and $\Phi_\mathbf{i} = [\Phi_{(\mathbf{i}-1)}, \Phi_{idx(i)}]$,

$\mathbf{C_i} = \begin{pmatrix} \mathbf{C}_{(\mathbf{i}-1)} & \mathbf{C}_{1:(i-1),i} \\ 0 & \mathbf{C}_{i,i} \end{pmatrix}$.

The third (*iii*) improvement is to discard negligible data and thereby eliminate unnecessary computations. In the *i*-th updated Gram matrix K^i, defined in (13), the diagonal elements k^i_{jj} represents the reconstruction error of the *j*-th data point with respect to the previous $(i-1)$ features. If k^i_{jj} is relatively small, then the previous $(i-1)$ features contain most of the information that would be acquired from this image vector. One can therefore optionally discard image points that satisfy $k^i_{jj} < \delta$, for some predetermined $\delta > 0$. In the following, we use a *cut-off threshold* that is useful when the data size is very large. It can also be used as a criterion to stop extracting features if one is unsure of the number of features that should be selected.

The entire algorithm of AKFA is summarized below:

Step 1: Compute the $n \times n$ Gram matrix $k_{ij} = k(x_i, x_j)$, where n is the number of input vectors. This part requires $O(n^2)$ operations.

Step 2: Let ℓ denote the number of features to be extracted. Initialize the $\ell \times \ell$ coefficient matrix $\mathbf{C}$ to 0, and $idx(\cdot)$ as an empty list which will ultimately store the indices of the selected image vectors. Initialize the threshold value $\delta = 0$ for the reconstruction error. The overall cost is $O(\ell^2)$.

Step 3: For $i = 1$ to ℓ repeat:

1. Using the *i*-th updated $\mathbf{K}^{\mathbf{i}}$ matrix, extract the *i*-th feature using (14). If $K_{jj}^{i} < \delta$, then discard *j*-th column and *j* -th row vector without calculating the projection variance. Use $idx(i)$ to store the index. This step requires $O(n^2)$ operations.
2. Update the coefficient matrix by using (16) and (17), which requires $O(i^2)$ operations.
3. Use (13) to obtain $\mathbf{K}^{i+1}$, an updated Gram matrix. Neglect all rows and columns that contain a diagonal element less than δ. This step requires $O(n^2)$ operations.

The total Computational complexity is increases to $O(\ell n^2)$ when no data is cut during updating in the proposed AKFA. If we increase δ, more data will be cut, and the total computational time will be decreased. If we chose to maintain good reconstruction accuracy, δ should be small. However, if we choose a shorter computation time, a large enough δ must be chosen. Since the number of data being cut at each step depends on δ and the data set, we consider an average situation. Suppose after extracting a feature, we only keep a fraction p of all possible directions, and let there be one point left after extracting ℓ features. That means $np^{\ell} = 1$, where *n* is the data size. It is equal to $p = n^{-1/\ell}$. The total computational time T is

$$T_{AKFA} = \sum_{k=0}^{\ell-1} (np^k)^2 = \frac{n^2 - 1}{1 - n^{-2/\ell}} \approx \frac{n^2}{1 - n^{-2/\ell}}. \tag{18}$$

when *n* is large. The computational time of SKFA is $T_{SKFA} = O(\ell^2 n^2)$. Therefore,

$$\frac{T_{SKFA}}{T_{AKFA}} = \ell^2 (1 - n^{-2/\ell}). \tag{19}$$

For example, when $\ell = 50$, data size $n = 3000$, T_{SKFA}/T_{AKFA} is about 684, so the speed of AKFA is 684 times faster than SKFA. Fig. 3.1 summarizes the comparison for computational complexity as

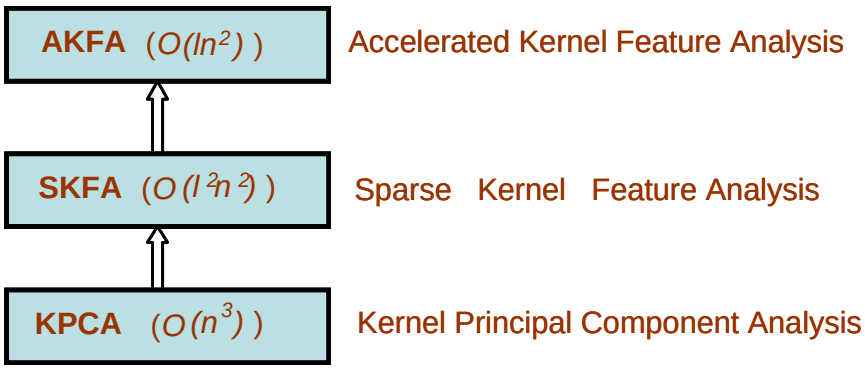

Figure 3.1 Computational Complexity

This figure illustrates the computational complexity of each algorithm discussed in the earlier sections. l is the number of features extracted for eigen dimension, and n is the data size.

CHAPTER 4 KERNEL ADAPTATION ANALYSIS

4.1 Kernels:

A kernel function provides a flexible and effective learning mechanism, and the choice of a kernel function should reflect prior knowledge about the problem at hand. However, it is often difficult for us to exploit the prior knowledge on patterns to choose a kernel function, and it is an open question how to choose the best kernel function for a given data set. According to no free lunch theorem [22] on machine learning, there is no superior kernel function in general, and the performance of a kernel function rather depends on applications.

Kernel K is a Hermitian and positive semi-definite matrix that computes the inner product between any finite sequences of inputs $x := \{x_j : j \in \mathrm{N}_n\}$ and is defined as: $K := (K(x_i, x_j) : i, j \in \mathrm{N}_n) = (\phi(x_i).\phi(x_j))$.

Some of the commonly used kernels are [23]:

The linear kernel: $K(x, x_i) = x^T x_i$ (20)

The polynomial kernel: $K(x, x_i) = (x^T x_i + offset)^d$ (21)

The Gaussian RBF kernel: $K(x, x_i) = \exp(-\|x - x_i\|^2 / 2\sigma^2)$ (22)

The Laplace RBF kernel: $K(x, x_i) = \exp(-\sigma\|x - x_i\|)$ (23)

The sigmoid kernel : $K(x, x_i) = \tanh(\beta_0 x^T x_i + \beta_1)$ (24)

The ANOVA RB kernel: $K(x, x_i) = \sum_{k=1}^{n} \exp(-\sigma(x^k - x_i^k)^2)^d$ (25)

Kernel selection is heavily dependant on the data specifics. For instance, the linear kernel is important in large sparse data vectors and it implements the simplest of all kernels whereas the Gaussian and Laplace RBF are general purpose kernels used when

prior knowledge about data is not available. Gaussian kernel avoids the sparse distribution caused by the high degree polynomial kernel in large feature space. The polynomial kernel is widely applied in image processing while ANOVA RB is usually reserved for regression tasks. The spline kernels are useful for continuous signal processing algorithms that involve Bspline inner-products or the convolution of several spline basis functions. For these reasons in our paper we are using kernels for Eq. (20) to Eq. (24) to from our data dependant composite kernel. The reason for this selection can be found in [24-27].

4.2 Kernel Selections:

In this method we exploit the idea of data dependant kernel that is presented in [29]to select most appropriate kernels for a given data so that we can use them for creating a Data Dependant Composite Kernel.

Let $\{x_i, y_i\}(i = 1,2,...,n)$ be n training samples of the given data, where $y_i = \{+1, -1\}$ represents the class labels of the samples. We engineer a data-dependent kernel to capture the relationship among the data in this classification task. This data-dependent composite kernel for r = 1,2,3,4,5 can be formulated as:

$$k_r(x_i, x_j) = q_r(x_i) q_r(x_j) p_r(x_i, x_j) \tag{26}$$

where, x Є R^d, $p_r(x_i, x_j)$ is one kernel among 5 chosen kernels and q(.), the factor function, takes the following form for r = 1,2,3,4,5;

$$q_r(x_i) = \alpha_{r0} + \sum_{m=1}^{n} \alpha_{rm} k_1(x_i, x_m) \tag{27}$$

in which $k_1(x_i, x_m) = \exp(-\|x_i - x_m\|^2 / 2\sigma^2)$, α_m is the combination coefficient. . Let us denote the vectors $\{q_r(x_1), q_r(x_2), \ldots\ldots q_r(x_n)\}^T$ and $\{\alpha_0, \alpha_1, \ldots\ldots \alpha_n\}_r^T$ by q_r and α_r (r=1,2,3,4,5) respectively where we have $q_r = K_0\ \alpha_r$ where K_0 is a $n \times (n+1)$ matrix given as follows:

$$K_0 = \begin{bmatrix} 1 & k_0(x_1, x_1) & \cdots & k_0(x_1, x_n) \\ 1 & k_0(x_2, x_1) & \cdots & k_0(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & k_0(x_n, x_1) & \cdots & k_0(x_n, x_n) \end{bmatrix}$$

(28)

Let the kernel matrices corresponding to $k(x_i, x_j)$, $p_r(x_i, x_j)$ be K, P_r. Therefore we can express data dependant kernel K as:

$$K_r = [q_r(x_i) q_r(x_j) p_r(x_i, x_j)]_{n \times n} \tag{29}$$

Defining Q_i as the diagonal matrix of elements $\{q_i(x_1), q_i(x_2), \ldots\ldots q_i(x_1)\}$ we can express Eq. (29) as:

$$K_r = Q_r P_r Q_r \tag{30}$$

This kernel model was first introduced in [28] and called "Conformal Transformation of a Kernel." In the following section, we perform kernel optimization based on the ideas presented in [30]

4.3 Finding the appropriate kernels for the data set:

The optimization of the data-dependent kernel in Eq. (26) is to set the value of combination coefficient vector α_r so that the class separability of the training data in mapped feature space is maximized. For this purpose, we use Fisher scalar as the objective function of our kernel optimization. Fisher scalar is used to measure the class separability *J* of the training data in the mapped feature space and is formulated as

$$J = \frac{tr(S_{br})}{tr(S_{wr})} \tag{31}$$

Where S_{b1}, S_{b2} represents "between-class scatter matrices" and S_{w1}, S_{w2} are the "within-class scatter matrices." Suppose that the training data are grouped according to their class

labels, i.e., the first n1 data belong to one class and the remaining n2 data belong to the other class ($n_1 + n_2 = n$). Then, the basic kernel matrix P_r (where r = 1, 2,3,4 represents the kernels from Eq. (20) to Eq. (24)) can be partitioned as follows

$$P_r = \begin{pmatrix} P_{11}^r & P_{12}^r \\ P_{21}^r & P_{22}^r \end{pmatrix} \tag{32}$$

Where the size of the sub- matrices $P_{11}^r, P_{12}^r, P_{21}^r, P_{22}^r$ are $n_1 \times n_1$, $n_1 \times n_2$, $n_2 \times n_1$, $n_2 \times n_2$ respectively. According to [12]the class seperability in Eq. (31) can be expressed as:

$$J(\alpha_r) = \frac{\alpha_r^T M_{0r} \alpha_r}{\alpha_r{}^T N_{0r} \alpha_r} \tag{33}$$

Where,

$$M_{0r} = K_0^T B_{0r} K_0^T \tag{34}$$

$$N_{0r} = K_0^T W_{0r} K_0^T \tag{35}$$

$$B_{0r} = \begin{pmatrix} \frac{1}{n_1} P_{11}^r & 0 \\ 0 & \frac{1}{n_2} P_{22}^r \end{pmatrix} - \frac{1}{n} P_r \tag{36}$$

$$W_{0r} = diag(p_{11}^r, p_{22}^r, ..., p_{nn}^r) - \begin{pmatrix} \frac{1}{n_1} P_{11}^r & 0 \\ 0 & \frac{1}{n_2} P_{22}^r \end{pmatrix} \tag{37}$$

To maximize $J(\alpha_r)$ in the Eq. (33), the standard gradient approach is followed. If matrix N_{0i} is nonsingular, the optimal α_r that maximizes the $J(\alpha_r)$ is the eigenvector corresponding to the maximum eigenvalue of the system,

$$M_{0r} \alpha_r = \lambda_r N_{0r} \alpha_r \tag{38}$$

The criterion for selecting the best kernel function is finding the kernel which produces largest Eigen value from (38), i.e.

$$\lambda_r^* = \arg\max_{\lambda}(N^{-1}M) \tag{39}$$

The idea behind is to choose the maximum Eigen vector alpha corresponding to maximum Eigen value that can maximize the $J_i(\alpha_i)$ will result in the optimum solution. Once we derive the eigen vectors i.e., the combination coefficients of all the 4 different kernels, we now proceed to construct q_r ($q_r = K_1\alpha_r(n)$) and Q_r to find the corresponding gram matrices of these kernels as according to the Eq. (30).

Once we have these gram matrices ready we can now find the optimum kernels for the given data set. To do this we have to arrange the Eigen values (that determined the combination coefficients) for all kernel functions and arrange them in the descending order. For example the following example shows one of the possible orders.

$$\lambda_1^* > \lambda_3^* > \lambda_4^* > \lambda_2^* > \lambda_5^* \tag{40}$$

Now first *p* kernels corresponding to fist *p* Eigen values will be used in the construction of a composite kernel that will yield optimum classification accuracy. This method is described in the following subsection.

4.4 Composite Kernel: Kernel Combinatory Optimization

In this section we propose composite kernel function which is defined as the weighted sum of the set of different optimized kernel functions. To obtain the optimum classification accuracy, we define the composite kernel as

$$K_{comp}(\rho) = \sum_{i=1}^{p} \rho_i Q_i P_i Q_i \tag{41}$$

Where the value of the composite coefficient ρ_i is a constant scalar and p is the number of kernels we intend to combine. Through this approach, the relative contribution of all the kernels to the model can be varied over the input space. And now instead of using K_r as Kernel matrix we will be using $K_{comp}(\rho)$ as Kernel Matrix and according to [31] this composite kernel matrix $K_{comp}(\rho)$ satisfies the Mercers condition.

Now the problem is to determine this composite coefficient $\hat{\rho}$ ($\hat{\rho} = \left[\rho_1, \rho_2, .., \rho_p\right]$) such that the classification performance is optimized. To solve this, we used the concept of "Kernel Alignment" to determine the best $\hat{\rho}$ that gives us optimum performance.

The "Alignment" measure was introduced by Cristianini et al. [32] in order to measure the adaptability of a kernel to the target data, and provide a practical objective for kernel optimization. The alignment measure is defined as a normalized Frobenius inner product between the kernel matrix and the target label matrix. The empirical alignment between kernel k_1 and kernel k_2 with respect to the training set S is given as:

$$A(k_1, k_2) = \frac{\langle K_1, K_2 \rangle}{\|K_1\|_F \|K_2\|_F} \tag{42}$$

Where K_i is the kernel matrix for the training set S using Kernel function k_i, $\|K_i\|_F = \sqrt{\langle K_i, K_i \rangle_F}$, $\langle K_i, K_j \rangle_F$ is the Frobenius inner product between K_i and K_j. If, X is the input space, y is the target vector then the training set can be defined as $S = \{(x_i, y_i) | x_i \in X, y_i \in \{+1, -1\}, i = 1,2,...,n\}$. Let $K_2 = yy^T$, then the empirical alignment between kernel k and target vector y is:

$$A(k, yy^T) = \frac{\langle K, yy^T \rangle_F}{\|K\|_F \|yy^T\|_F} = \frac{y^T K y}{n\|K\|_F} \tag{43}$$

It has been shown that if a kernel is well aligned with the target information, there exists a separation of the data with a low bound on the generalization error [32]. Thus, we can optimize the kernel alignment based on training set information to improve the generalization performance on the test set. Let us consider the combination of kernel functions corresponding to Eq. (41) as:

$$k(\rho) = \sum_{i=1}^{p} \rho_i k_i \tag{44}$$

Where, the kernels $k_i, i = 1,2,..., p$ are known in advance. Our purpose is to tune ρ to maximize $A(\rho, k, yy')$, the empirical alignment between $k(\rho)$ and the target vector y. Hence,

$$\hat{\rho} = \arg_{\rho} \max\left(A\left(\rho, k, yy^T\right)\right) \tag{45}$$

$$= \arg_{\rho} \max\left(\frac{\left\langle \sum_i \rho_i K_i, yy^T \right\rangle}{n\sqrt{\left\langle \sum_i \rho_i K_i \right\rangle, \left\langle \sum_j \rho_j K_j \right\rangle}} \right) \tag{46}$$

$$= \arg_{\rho} \max\left(\frac{\sum_i \rho_i \left\langle K_i, yy' \right\rangle}{n\sqrt{\sum_{i,j} \rho_i, \rho_j \left\langle K_i, K_j \right\rangle}} \right) \tag{47}$$

$$= \arg_{\rho} \max \left(\frac{\left(\sum_{i} \rho_i u_i \right)^2}{n^2 \sum_{i,j} \rho_i \rho_j v_{ij}} \right) \tag{48}$$

$$= \arg_{\rho} \max \left(\frac{1}{n^2} \cdot \frac{\rho^T U \rho}{\rho^T V \rho} \right) \tag{49}$$

Where $u_i = \langle K_i, yy' \rangle, U_{ij} = u_i u_j, V_{ij} = v_{ij} = \langle K_i, K_j \rangle$. Let the generalized Raleigh coefficient be:

$$J(\rho) = \frac{\rho^T U \rho}{\rho^T V \rho} \tag{50}$$

Therefore we can obtain the value of $\hat{\rho}$ by solving the generalized eigen value problem

$$U\rho = \delta V \rho \tag{51}$$

where δ denotes the eigen values. Once we find this optimum composite coefficient $\hat{\rho}$ which will be the Eigen vector corresponding to maximum Eigen value δ, we now compute the composite data dependant kernel matrix $K_{comp}(\rho)$ according to Eq. (41). In this way we find the Kernel Gram Matrix for the offline data.

4.5 Determining the optimum number of kernels

In this section we develop a method to determine the optimum number of kernels that are required to form a composite kernel, starting form single kernel to 5 kernels: Linear, Polynomial, Gaussian, Laplace and Sigmoid Kernel (or Hyperbolic Tangential Kernel). We select the optimum number of kernels p as follows:

$$p = \min(\omega, \xi, \pi) \tag{52}$$

where

ω : Time complexity dependence parameter

ξ : Eigen value disparity

π : Reconstruction error dependence parameter.

They are explained in detail as follows:

4.5.1 Time complexity Dependence Parameter 'ω ':

The more the number of kernels, the greater is the number of calculations required and hence the time complexity.

The time complexity of the data dependent kernel (Section 4.2) : $O(n^2)$

Time Complexity of Coefficient computation: $O(rn)$, r is the number of kernels we have.

The complexity of KPCA algorithm : $O(n^3)$; n is the data size

The complexity of AKFA algorithm : $O(l\,n^2)$; l is the no. of features extracted for eigen dimension;

Therefore to perform classification with KPCA we need: $O(n^3)+O(n^2)+O(rn)$ and to perform classification with AKFA we need: $O(n^2 l)+O(n^2)+O(rn)$. Note that the minimum time complexity depends on AKFA, not KPCA, thus we only consider $O(n^2 l)+O(n^2)+O(rn)$.

This increases drastically with the size of data n as well as the number of kernels r. Since the data size is constant for all the kernels, $O(rn)$ is the difference factor. This value is dependant both on r (no.of kernels) as well as n (the size of the data). Therefore we introduce'ω' the Time complexity dependence parameter to determine the optimum number of kernels that would limit the unnecessary computation. Initially we set the value of ω to 0 and increment it each time when the condition $rn \le \varepsilon$ is satisfied. Due to the huge data sizes of our data bases we have empirically set the value of ε to 4000.

4.5.2. Eigen value disparity ξ :

It is better to limit the number of kernels unless their contribution is significant, *i.e*, their class seperability is good. We will consider a combination of kernels whose eigen values are very close to the most dominant eigen value. To determine the kernels that can be combined we introduce ξ , the measure of disparity between the dominant kernel and the other kernels. The eigen values of the Kernels (mentioned in V.A) are: $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$

To determine the most dominant kernel, let us define their descending order as follows:

$$\lambda_{l\arg e1}, \lambda_{l\arg e2}, \lambda_{l\arg e3}, \lambda_{l\arg e4}, \lambda_{l\arg e5}$$

The disparity measures can be measured as follows:

$$\xi_1 = \lambda^1 - \lambda^2, \quad \xi_2 = \lambda^1 - \lambda^3, \quad \xi_3 = \lambda^1 - \lambda^4, \quad \xi_4 = \lambda^1 - \lambda^5$$

Now, the Eigen value disparity factor can be computed by initially setting it to 1 and incrementing it whenever the criteria $\xi_j \le \xi_1 + 0.2\xi_1$ (j =1,2,3,4) is met. This criteria indicates that we are considering all the kernels which produce ξ_j that is in 20% deviation of the least eigen value disparity ξ_1.

After obtaining ω, ξ , we find the value of p from Eq. (52) to determine the number of kernels to be used in the computation of Data Dependant Composite kernel. After developing this Kernel, we use it in the eq. (6) of Section 3.1 and 3.3 and proceed with either KPCA or AKFA algorithm for creating a higher dimensional feature space.

CHAPTER 5 EXPERIMENTAL RESULTS OF KERNEL ADAPTATION ANALYSIS

The performance of Data Dependant Composite Kernel methodology is evaluated using 8 different data sets: 4 different Colon Cancer data sets, Breast Cancer, Lung Cancer, Lymphoma and Prostate Cancer. The sources of each Data set and their composition is discussed in section 5.1. The eigen values of each Data dependant Kernel matrices of each data set generated from 5 different kernels are listed in Section 5.2. Determination of optimum number of data dependant kernel matrices and their respective linear combination is discussed in Section 5.3 and finally section 5.4 deals with the classification accuracy, mean square reconstruction error of our proposed method for all the data sets.

5.1 Data Sets

There are 4 colon cancer data sets. Subsections 5.1.1, 5.1.2 describe the acquisition and composition of Colon Cancer data sets.

5.1.1 Colon Cancer Data Set 1

This data set is obtained by evaluating CT image data sets of colonic polyps comprised of Positives (P) and Negatives (N) (i.e., true polyps and false polyps respectively) detected by our CAD system [5]. We obtained studies of 146 patients who had undergone a colon-cleansing regimen in preparation for same-day optical colonoscopy. Each patient was scanned in both supine and prone positions, resulting in a total of 292 CT studies. Helical single-slice and multi-slice CT scanners were used, with collimations of 1.25 - 5.0 mm, reconstruction intervals of 1.0 – 5.0 mm, X-ray tube currents of 50 – 260 mA and voltages of 120 – 140 kVp. In-plane voxel sizes were 0.51– 0.94 mm, and the CT image matrix size was 512 x 512. Out of 146 patients, there were 108 normal cases and

38 abnormal cases with a total of 61 colonoscopy-confirmed polyps larger than 6 mm. Twenty-eight polyps were 6-9 mm and 33 polyps were larger than 10 mm (including 7 lesions larger than 30 mm

The volumes of interest (VOIs) representing each polyp candidate have been calculated as follows. The CAD scheme provided a segmented region for each candidate. The center of the VOI was placed at the center of mass of the region. The size of the VOI was chosen so that the entire region was covered. Finally, the VOI was resampled to 12×12×12 voxels to build the Colon Cancer data set1 that consists of 39 true polyps and 149 false polyps. In the following tables 'P' represents Positive i.e., true polyp and 'N' represents Negative i.e., false polyp.

Table 5.1
Data Distribution in Colon Cancer Data Set 1

			Proportion from the total data set	Number of vectors	Data size
Arrangement 1	Training Set	P	80.00%	31	148
		N	78.30%	117	
	Test Set	P	20%	8	40
		N	21.70%	32	

5.1.2 Colon Cancer Data Sets 2, 3 and 4

This Data Base is obtained by evaluating the performance of our CAD scheme. We collected 43 CTC cases retrospectively at the University of Chicago Hospitals. These cases were obtained by use of a helical CT scanner (GE CTi 9800, General Electric Medical Systems, Milwaukee WI) with collimation of 5 mm, a pitch of 1.5–1.7, and reconstruction intervals of 1.5–2.5 mm. The current was 100 mA with 120 kVp. A 180 linear interpolation and the standard reconstruction algorithm were used. The colon was cleansed with a standard pre colonoscopy cleansing method and insufflated with room air when the CT scanning was performed. Each patient was scanned in both supine and prone positions. Eleven of the cases contained a total of 12 colonoscopy proven polyps of at least 5 mm in diameter. This size is the lower limit of the size range for the polyps that

are considered to be clinically significant [3]. Nine polyps measured between 5 and 10 mm; the other three measured 12, 25, and 30 mm. Two experienced radiologists examined the locations of these polyps in the CTC data sets. We had a total of 21 visible polyps. Because two of these visible polyps were in a single data set, we had 20 CTC data sets, among 86 data sets, with at least one visible polyp.

Each CTC data set covered the entire region of the abdomen, from diaphragm to rectum, and consisted of 150–300 CT images with a matrix size of 512x 512. After the linear interpolation along the axial direction, the dimension of the resulting isotropic volumes contained between 500 and700 voxels, and the physical resolution of these volumes was 0.5–0.75 mm/voxel.

The volumes of interest (VOIs) representing each polyp candidate have been calculated same as mentioned above except that the VOI was resampled to 16×16×16 voxels. The VOIs so computed comprise the data set DB1, a sample of which is shown in Figure 1. There were a total of 131 true polyps (some of the larger lesions had multiple detections) and 8008 false polyps. We divided this huge Data base into 3 small data bases Colon Cancer Data Set2, Colon Cancer Data Set3 and Colon Cancer Data Set4.

Table 5.2
Data Distribution in Colon Cancer Data Set2 (U of Chicago)

			Proportion from the total data set	Number of vectors	Data size
Arrangement 2	Training Set	P	80.00%	66	766
		N	70.00%	700	
	Test Set	P	20%	16	316
		N	30%	300	

Table 5.3
Data Distribution in Colon Cancer Data Set3 (BID)

			Proportion from the total data set	Number of vectors	Data size
Arrangement 3	Training Set	P	80.00%	22	1012
		N	70.00%	990	
	Test Set	P	20%	6	431
		N	30%	425	

Table 5.4
Data Distribution in Colon Cancer Data Set3 (North Western University)

			Proportion from the total data set	Number of vectors	Data size
Arrangement 4	Training Set	P	80.00%	17	1817
		N	60.00%	1800	
	Test Set	P	20%	4	1204
		N	40%	1200	

5.1.3 Breast Cancer

This data set contains data on 189 women, 64 of which were treated with tamoxifen, with primary operable invasive breast cancer. More information on this data set can be found in [33].

Table 5.5
Data Distribution in Breast Cancer Data Set

			Proportion from the total data set	Number of vectors	Data size
Arrangement 5	Training Set	P	80%	51	126
		N	60%	75	
	Test Set	P	20%	13	63
		N	40%	50	

5.1.4 Lung Cancer

This data set is available at http://www.broadinstitute.org/cgi-bin/cancer/datasets.cgi. It contains It contains 160 tissue samples, of which 139 are of class '0' and the remaining are of class'2'. Each sample is represented by the expression levels of 1000 genes.

Table 5.6
Data Distribution in Lung Cancer Data Set

			Proportion from the total data set	Number of vectors	Data Size
Arrangement7	Training Set	P	70%	15	126
		N	80%	111	
	Test Set	P	30%	6	34
		N	20%	28	

5.1.5 Lymphoma Cancer

The data set is available at http://www.broad.mit.edu/mpr/lymphoma. It contains 77 tissue samples, of which 58 are diffuse large B-cell lymphomas (DLBCL) and rest of them are follicular lymphomas (FL). Detailed information about this data set can be found in [34].

Table 5.7
Data Distribution in Lymphoma Data Set

			Proportion from the total data set	Number of vectors	Data Size
Arrangement 6	Training Set	P	85%	16	62
		N	79%	46	
	Test Set	P	15%	3	15
		N	21%	12	

5.1.6 Prostate Cancer

This data is from http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE6919. It contains Prostate cancer data collected from 308 patients, 75 of which have Metastatic Prostate Tumor and the rest of the cases were normal. More information on this data set can be found in [35], [36].

Table 5.8
Data Distribution in Prostate Cancer Data Set

			Proportion from the total data set	Number of vectors	Data Size
Arrangement7	Training Set	P	85%	64	250
		N	80%	186	
	Test Set	P	15%	11	58
		N	20%	47	

5.2. Kernel Selection

To select the kernel that will best fit the data, we perform kernel selection according to the method proposed in chapter 4, section 4.2. In this, we first compute 5 data dependant kernels, each resulting form a different kernel function and then compute the eigen values of the combination vector. The larger the eigen value the greater is the class seperability measure J in the equation (31). Hence we evaluated the algorithm described in section V.A on all the data sets mentioned in VI.A to determine the eigen values of all the 5 kernels for each and every data set and listed them in the table 5.9.

Table 5.9
Eigen values of all Kernels and their parameters for each data set.

Data Sets	λ and parameters for Linear kernel (Eq 20)	λ and parameters for Polynomial Kernel (Eq 21)	λ and parameters for Sigmoid kernel (Eq 22)	λ and parameters for Gaussian RBF kernel (Eq 23)	λ and parameters for Laplace RBF Kernel (Eq 24)
Colon Cancer DataSet1	λ =13.28	λ =11.54 d = 1, offset = 1	λ =3.02 β_0 =2, β_1 =1.75	λ =**16.82** σ =4.00	λ =7.87 σ =0.1
Colon Cancer Dataset 2	λ =75.43	λ =84.07 d = 1,offset = 4	λ =64.49 β_0 =1, β_1 =2.5	λ =**139.96** σ =5.65	λ =40.37 σ =1.5
Colon Cancer Dataset 3	λ =100.72	λ =106.52 d = 1,offset = 1	λ = 53.23 β_0 =2, β_1 =3	λ =**137.74** σ =4.47	λ =80.67 σ =1.5
Colon Cancer Dataset 4	λ =148.69	λ =166.44 d = 1,offset = 1	λ =142.32 β_0 =1, β_1 =2	λ =**192.14** σ =4.58	λ =34.99 σ =3
Breast Cancer	λ =22.85	λ =20.43 d =1.2,offset =1	λ = 23.21 β_0 =0.75, β_1 =1	λ =**64.38** σ =4.47	λ = 56.85 σ =1
Lung Cancer	λ =36.72	λ = 47.49 d = 1.2,offset=4	λ =29.23 β_0 =4, β_1 =2.5	λ = **54.60** σ =3.87	λ = 38.74 σ =2.4
Lympho ma	λ =19.71	λ =37.50 d = 1.5,offset=2	λ =23.64 β_0 =1.5, β_1 =2	λ =**42.13** σ =2.82	λ = 35.37 σ =2
Prostate cancer	λ =50.93	λ = 48.82 d = 1,offset = 1	λ = 43.10 β_0 =0.5, β_1 =0.5	λ =**53.98** σ =4.47	λ = 40.33 σ =1.5

The parameters of all the kernels described from Eq (20) to Eq (24) and computed their eigen values for all the 8 data sets are tabulated above. We determine the optimum kernel depending on the eigen value that will produce maximum seperability i.e., the kernel that yields highest eigen value for a given data set. After arranging the eigen values for each data set in the descending order we selected the kernel corresponding to the largest eigen value as the optimum kernel. (The largest eigen value for each data set is highlighted in the table above.) After evaluating the algorithm for all the data sets, we observed that the RBF kernel gives the maximum eigen value of all the 5 kernels.

5.3. Combination of kernels

In order to perform the kernel combination described in section 4.3, we used the evaluation explained in Section 4.4 to determine the number of kernels for all the 8 data sets that can be combined to yield best results. We used equation (52) to determine the optimum number of kernels that can be used for the Kernel Combination. The following table illustrates the method to determine the optimum number of kernels that can be used for forming the data dependant composite kernel.

Table 5.10
Number of Kernel Selected

Data Sets	ω	ξ	p No. of kernel functions selected Eq(52)
Colon Cancer Data Set1	5	2	2
Colon Cancer Data set 2	5	2	2
Colon Cancer Data set 3	3	2	2
Colon Cancer Data set 4	2	2	2
Breast Cancer	5	2	2
Lung Cancer	5	2	2
Lymphoma	5	2	2
Prostate cancer	5	2	2

From the results it is clear that the eigen value disparity ξ is the most dominant factor in determining the number of kernels p for all the data sets. The time complexity dependence parameter played an important role in the case of Colon Cancer data sets 3 and 4. Depending on the minimum value of the parameters ω and ξ evaluated for all the data sets, the optimum number of kernels p for all the data sets is determined to be 2.

There fore according to method followed in 4.4, we combine 2 most dominant data dependant kernels to form a Data Dependant Composite kernel that has the potential to yield optimum classification accuracy. The kernels used for creating data dependant composite kernel and their linear combinations are listed in the following table.

Table 5.11
The value of $\hat{\rho}$ for all the data sets

Data Sets	Two selected kernels	Composite Coefficients
Colon Cancer Data set 1	RBF and Linear	$\hat{\rho}_1$=0.9852, $\hat{\rho}_2$=0.1527
Colon Cancer Data set 2	RBF and Polynomial	$\hat{\rho}_1$=0.672, $\hat{\rho}_2$=0.1582
Colon Cancer Data set 3	RBF and Polynomial	$\hat{\rho}_1$=0.992, $\hat{\rho}_2$=0.1204
Colon Cancer Data set 4	RBF and Polynomial	$\hat{\rho}_1$=0.9775, $\hat{\rho}_2$=0.1375
Breast Cancer	RBF and Laplace	$\hat{\rho}_1$= 0.8573 , $\hat{\rho}_2$= 0.1386
Lung Cancer	RBF and Laplace	$\hat{\rho}_1$=0.97930, $\hat{\rho}_2$= 0.1261
Lymphoma	RBF and Polynomial	$\hat{\rho}_1$=0.9903, $\hat{\rho}_2$=0.2082
Prostate cancer	RBF and Linear	$\hat{\rho}_1$=0.9756, $\hat{\rho}_2$=0.1219

For almost all the data sets, RBF kernel is the most appropriate kernel because of it is naturally normalized. Polynomial kernel is the second dominant kernel for 4 data sets (Colon Cancer data sets 2, 3 and 4, Lymphoma data set). Laplace Kernels was second best kernels for Breast Cancer and Lung Cancer and Linear kernel was for Colon Cancer data set 1 and Prostate Cancer. Therefore the resulting Data dependant composite kernel

will take the form of $K_{comp}(\rho) = \rho_1 *$ (Most Dominant Kernel) + $\rho_2 *$ (Second Dominant Kernel)

5.4. Classification Accuracy and Mean Square Reconstruction Error of Data Dependant Composite Kernel.

5.4.1. Classification Accuracy

In order to analyze how the feature extraction methods affect classification performance of polyp candidates, we used the *k*-nearest neighborhood classifier on the image vectors in the reduced eigenspace. We evaluated the performance of the classifier by applying it to the feature spaces obtained by KPCA and AKFA with both selected single kernel as well as Data Dependant Composite Kernel for all the 8 data sets. 6 nearest neighbors were used for the classification purpose. The classification accuracy was calculated as (TP+TN)/(TP+TN+FN+FP). The results are tabulated as follows.

Table 5.12
Classification accuracy for each feature extraction algorithm against the *6* nearest neighbors for the Composite Kernel.

Data Sets	Classification accuracy of KPCA with single kernel	Classification accuracy of KPCA with data dependant composite Kernel	Classification accuracy of AKFA with single kernel	Classification accuracy of AKFA with data dependant composite Kernel
Colon Cancer Data Set1	97.50	97.50	92.5	95.00
Colon Cancer Data Set2	94.94	94.94	94.94	94.94
Colon Cancer Data Set3	98.61	98.61	98.61	98.61
Colon Cancer Data Set4	99.67	99.67	99.67	99.67
Breast Cancer	87.5	95.24	95.21	96.83
Lung Cancer	91.18	94.12	91.18	94.12
Lymphoma	87.5	93.75	87.5	93.75

Prostate cancer	87.96	96.55	89.66	91.38

The results from the table clearly demonstrate that the classification accuracy of the Data Dependant Composite kernel is better than that of the single kernel for both KPCA and AKFA in Colon cancer data set1, Breast cancer, Lung Cancer, Lymphoma and Prostate Cancer; whereas in the case of Colon cancer data sets 2, 3, 4 because of the huge size of the data, the classification accuracy is the same in all the three cases. From the table it is also clear that in most of the cases with the composite kernel, the Classification performance of AKFA is equally good as that of KPCA.

5.4.2. Mean Square Reconstruction Error (MSE)

The reconstruction error results have been evaluated for KPCA using Eq. (15), and for AKFA using Eq. (27) with the optimum kernel (RBF) selected form the above table. We evaluated the reconstruction errors of KPCA and AKFA for all the data sets both using a single kernel (RBF Kernel) and Data dependant composite kernel. The following table summarizes these results.

Table 5.13
Mean Square Reconstruction Error

Data Sets	KPCA Mean Square Error with RBF Kernel (%)	AKFA Mean Square Error with RBF Kernel (%)	KPCA Mean Square Error with Data Dependant Composite Kernel (%)	AKFA Mean Square Error with Data Dependant Composite Kernel (%)
Colon Cancer Data Set1	6.86	10.86	4.18	5.69
Colon Cancer Data set 2	12.39	18.11	10.01	15.26
Colon Cancer Data set 3	14.30	20.59	5.23	7.70
Colon Cancer Data set 4	12.48	18.14	10.50	14.16

Breast Cancer	6.05	10.10	4.01	6.56
Lung Cancer	4.92	7.30	2.43	3.67
Lymphoma	3.27	3.87	2.01	2.29
Prostate cancer	10.33	13.83	4.45	8.26

The reconstruction error of KPCA is less than that of the reconstruction error of AKFA. The difference in the reconstruction error between KPCA and AKFA increased as the size of the data sets increased. This could be due to the heterogeneous nature of the data sets. The Lymphoma data set produced the least mean square error where as the Colon Cancer data set 3 produced the largest mean square error for KPCA and AKFA in the case of both single kernel and data dependant composite kernel.

The smaller reconstruction error results of mse with data dependant composite kernel from the above table made evident that the reconstruction ability of kernel optimized KPCA and AKFA gives enhanced performance to that of single kernel KPCA and AKFA. Moreover, the improvement in the reconstruction error performance is greater in the case of AKFA when compared to that of KPCA. The best improvement of error performance is observed in the case of Colon cancer data set3. This successfully shows that the Composite kernel produces small reconstruction error.

5.4.3. Computational Time

We evaluated the computational efficiency of the proposed method by comparing its run time with the other two methods for different data sizes. The algorithms have been implemented in Matlab 7.0.1 (R14) using the Statistical Pattern Recognition Toolbox for the Gram matrix calculation and kernel projection. The processor was a 2.5 GHz Intel® Core™2 Duo T9300 with 3 MB of RAM. Run time was determined using the *cputime* command.

Table 5.14
Computation Time (in Sec) of KPCA vs AKFA

Data Sets	Computational time (sec)	
	KPCA	AKFA
Colon Cancer Data Set1	0.2656	0.2184
Colon Cancer Data Set2	2.8906	1.8750
Colon Cancer Data Set3	6.8281	3.2969
Colon Cancer Data Set4	31.9219	11.2344
Breast Cancer	0.2656	0.2188
Lung Cancer	0.1250	0.0625
Lymphoma	0.0781	0.0469
Prostate cancer	1.7031	1.1094

The following figure 5 illustrates the increase in the computational time of both KPCA as well as AKFA with increase in the data size. For this we listed the sizes of all the data sets in the ascending order from Lymphoma (77) to Colon Cancer Data set 4 (4021) on the X-axis vs respective computational times on the Y-Axis. The Blue curve indicates the computational time for the KPCA where as the Green curve increases the computational time for AKFA for all the 8 data sets arranged in ascending order of their sizes.

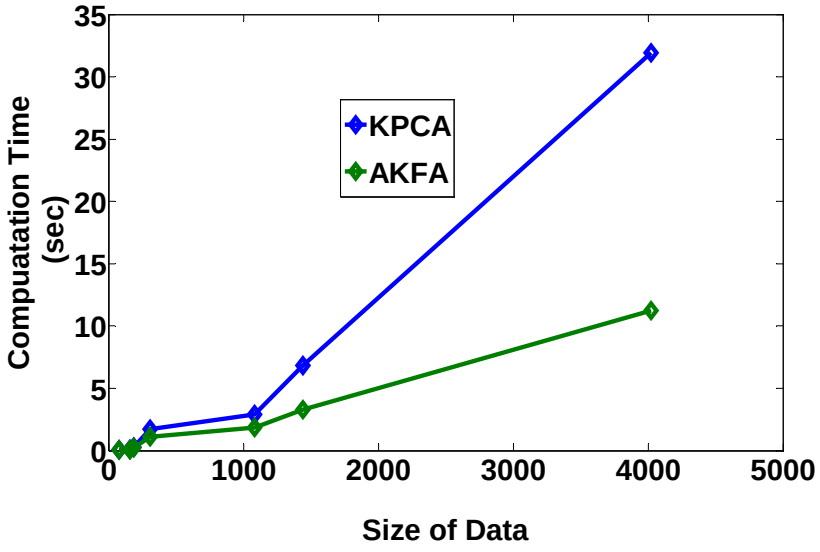

Figure 5.1 Size of Data vs Computation Time for KPCA and AKFA

For each algorithm, computation time increases with increasing training data size (n), as expected. AKFA requires the computation of a Gram matrix whose size increases as the data size increases. The results from the table clearly indicate that AKFA is faster than KPCA. We noticed that the decrease in computation time for AKFA compared to KPCA was relatively small, implying that the use of AKFA on a smaller training data set does not yield much advantage over KPCA. However, as the data size increases the computational gain for AKFA is much larger than that of KPCA.

5.5. Receiver Operating Characteristic Curves

A **receiver operating characteristic** (**ROC**), or simply **ROC curve**, is a graphical plot of the sensitivity, or true positives, vs. (1 −specificity), or false positives, for a binary classifier system as its discrimination threshold is varied. The ROC can also be represented equivalently by plotting the fraction of true positives (TPR = true positive rate) vs. the fraction of false positives (FPR = false positive rate). It is also known as a Relative Operating Characteristic curve, because it is a comparison of two operating characteristics (TPR & FPR) as the criterion changes.

For our polyp detection application, the available training sets have a very small number of positives (polyps). Therefore, it is unfeasible to generate meaningful ROCs for a test set which is a subset of these training sets. We decided to generate several ROCs for the training sets themselves, in an attempt to assess the classifiability of the data. The methods we used to generate the ROCs are described below.

5.5.1. Dense KNN:

The simplest and most intuitively understandable algorithm for data analysis and mapping is the k-Nearest Neighbours algorithm. In pattern recognition, the ***k*-nearest neighbors algorithm** (*k*-NN) is a method for classifying objects based on closest training examples in the feature space. Often this method is used for quick visualization (a preview) of data or as a benchmark tool for the comparison with other models.

k-NN is an example of the so-called "lazy learning" algorithms. With such approach, the function is modeled locally and all computations are made directly during the prediction step. Therefore there is no actual training phase – all training examples are just stored in the memory for further predictions. To make a prediction at some point in a feature space, which can be a high dimensional and not only geographical, one finds the first *k* nearest training points, according to some predefined distance measure. The prediction is a mean over the values of its *k* neighbours. To run the *k* nearest neighbours algorithm, one needs to define the distance measure and the number of neighbours to be used. In general, Minkowski *p*-norm distance can be used:

$$d_s(x, y) = \left(\sum_i |x_i - y_i|^s \right)^{1/s}$$

where *s* is a parameter equal or greater than one. So, with $s = 1$ it is a Manhattan (or city block) distance, with $s = 2$ – Euclidean, and $s = \infty$ corresponds to the infinity norm distance, that is, *max* |*xi* - *yi*|. Let us remind that the selection of *s* parameter can have important consequences on the corresponding results. The most widely used, especially because of its straightforward interpretation, is the Euclidean distance. The special case of the k-NN algorithms is when $k = 1$. It is called the nearest neighbor algorithm and corresponds to Voronoi polygons while working with geographical data. For different datasets the optimal number of neighbors - parameter *k*- is different.

In the experiment analysis, 9 nearest neighbors were used for both DB2 and DB3 data sets. Both the data sets were of 100 dimensions. By using this dense KNN algorithm, we have obtained the following ROC curves for the 2 large data sets DB2 and DB3. The dense KNN algorithm in [49], attacks the problem by building two ball trees: one ball tree is built from all the positive points in the dataset, another ball tree is built from all negative points. Then it classifies a new target point t and it computes q, the number of k nearest neighbors of t that are in the positive class, in the following fashion:

• Step 1. Find the k nearest positive class neighbors of t (and their distances to t) using conventional ball tree search.

• Step 2. Do sufficient search of the negative tree to prove that the number of positive data points among k nearest neighbor is q for some value of q. In this way the dense

KNN algoritm differentiates between true positives and false positives.

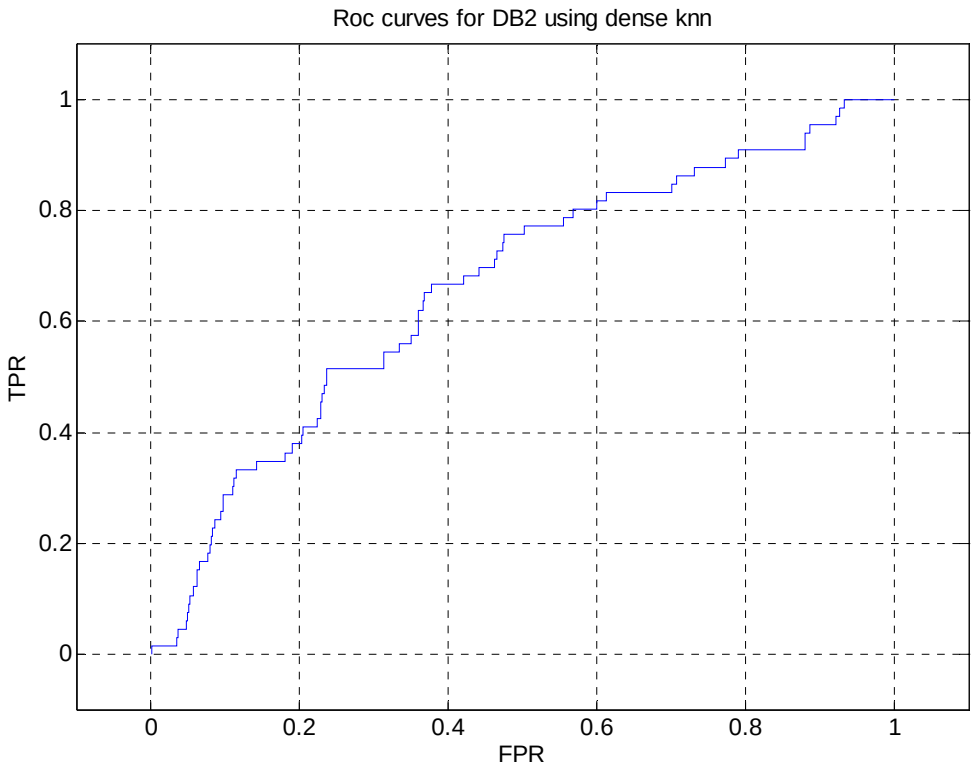

Figure 5.2 ROC curves for DB2 using Dense KNN

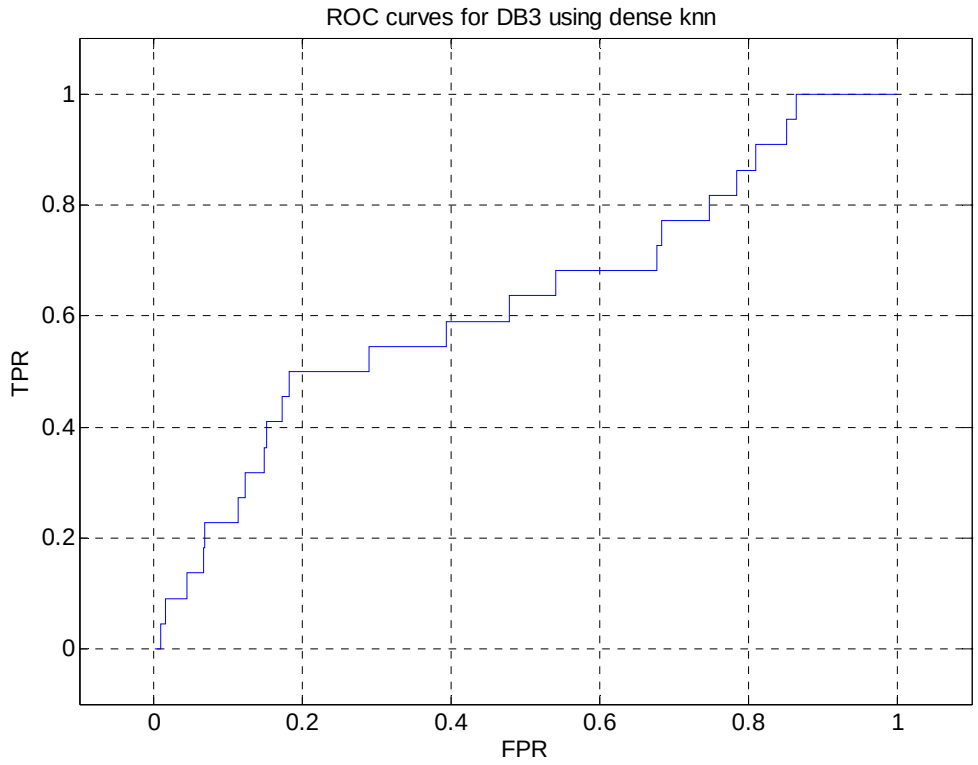

Figure 5.3 ROC curves for DB3 using Dense KNN

5.5.2. Linear classifier

In the field of machine learning, the goal of classification is to use an object's characteristics to identify which class (or group) it belongs to. A **linear classifier** achieves this by making a classification decision based on the value of a linear combination of the characteristics. An object's characteristics are also known as feature values and are typically presented to the machine in a vector called a feature vector. If the input feature vector to the classifier is a real vector $\vec{x}$, then the output score is

$$y = f\left(\vec{w}.\vec{x}\right) = f\left(\sum_j w_j x_j\right),$$

where $\vec{w}$ is a real vector of weights and f is a function that converts the dot product of the two vectors into the desired output. (In other words, $\vec{w}$ is a one-form or linear functional mapping $\vec{x}$ onto **R**.) The weight vector $\vec{w}$ is learned from a set of labeled training samples. Often f is a simple function that maps all values above a certain threshold called "bias" or "b" to the first class and all other values to the second class. A more complex f might give the probability that an item belongs to a certain class.

For a two-class classification problem, one can visualize the operation of a linear classifier as splitting a high-dimensional input space with a hyperplane: all points on one side of the hyperplane are classified as "yes", while the others are classified as "no". A linear classifier is often used in situations where the speed of classification is an issue, since it is often the fastest classifier, especially when $\vec{x}$ is sparse. Also, linear classifiers often work very well when the number of dimensions in $\vec{x}$ is large, as in document classification, where each element in $\vec{x}$ is typically the number of occurrences of a word in a document.

In our work, we obtain different ROC curves by varying the bias 'b' so as to produce a continuous curve that catches all possible combinations of true positives and false positives for a given data set. The ROC curves are evaluated for dimensions 125,100 and 50 in the case of DB2. In the case of DB3, since we have too few positives (22 positives vs. 990 negatives), any dimensions above 60 gave a perfect separation. So, we evaluated the performance of the "Linear Classifier" on the training set for 58, 50 and 30 dimensions in the case of DB3. The ROC curves produced by linear classifier for the 2 data sets are as follows:

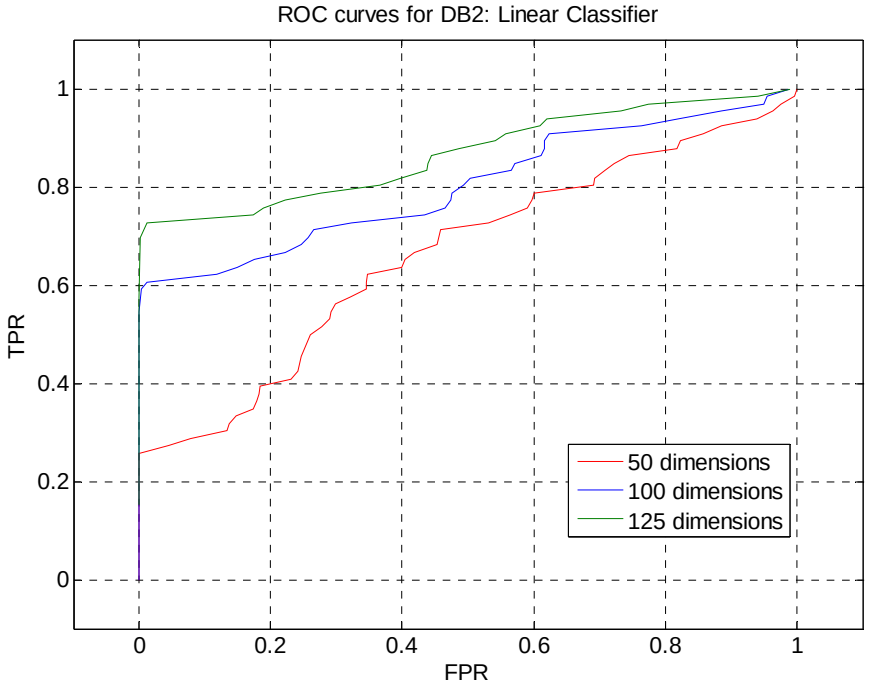

Figure 5.4 ROC curves for DB2 using Linear Classifier

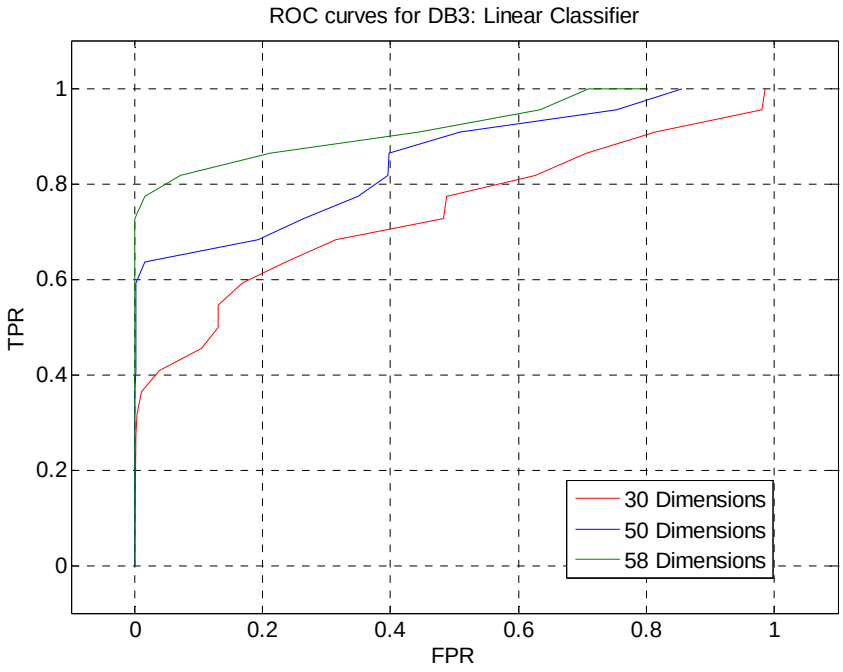

Figure 5.5 ROC curves for DB3 using Linear Classifier

5.5.3. Elliptic classifier

This kind of classifier models the positives in the data set as a Gaussian distribution with mean '**μ**' and covariance **'Σ'.** We find the mean and covariance of the positives in the dataset and construct an optimum ellipsoid in n –dimensions that fit the maximum number of positives in it. All the data points inside the ellipsoid are classified as positives and all the data points outside this ellipsoid are classified as negatives by this elliptic classifier. The output score is:

$$y = (x-\mu)^T \sum^{-1} (x-\mu)$$

If y is less than 1 (a threshold value) then the corresponding data point is inside the ellipsoid. Otherwise, it is outside the ellipsoid. We obtain the ROC curves for the 2 data sets by varying this threshold value. Note that the output score is the Mahalanobis distance between **x** and **μ**.

Both in the case of DB2 and DB3, there was a perfect separation in the dimensions greater than 50. So we evaluated data set DB2 for 50, 30, 20 dimensions and DB3 for 15, 12, 10 dimensions. The resulting ROC curves obtained are as follows:

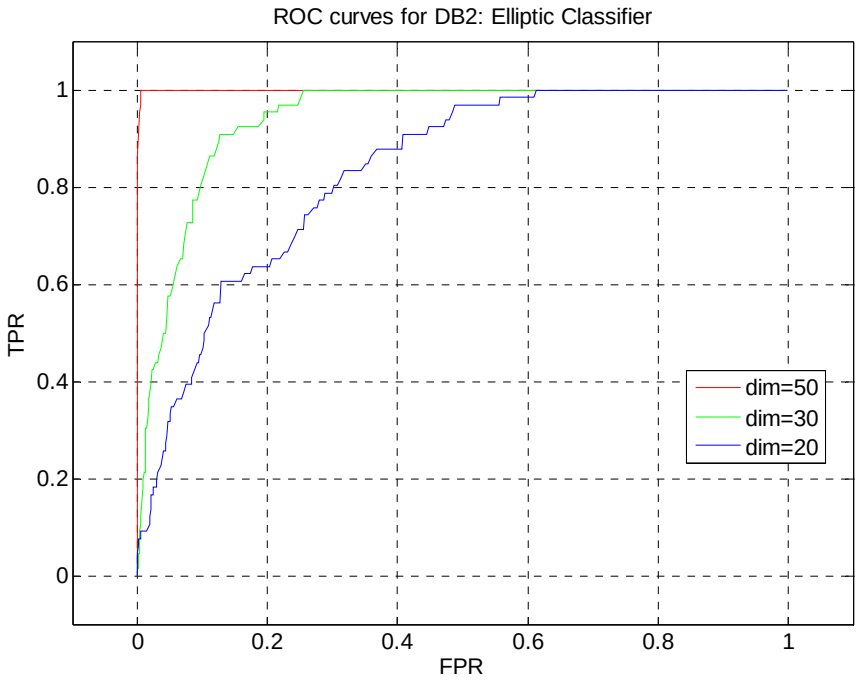

Figure 5.6 ROC curves for DB2 using Elliptic Classifier

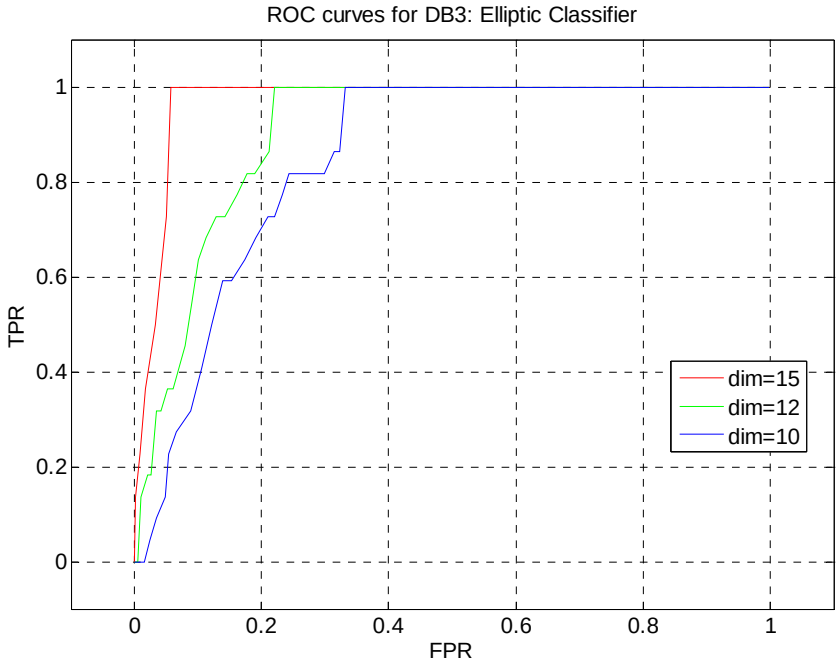

Figure 5.7 ROC curves for DB3 using Elliptic Classifier

Even though the performance of the elliptic classifier seem to be good, it reality this is not true

because, the ratio of number of positives in both the data sets to that of the dimensions being considered is very small. In such situation, the resulting ellipsoid may fail to include a positive data when it is applied on the test set data points. In order to develop a highly reliable model, we therefore need more positives so as to have a thick data distribution in large dimensions.

5.5.4. Extended-sets:

Since the number of positives in both the data sets are small (DB2 – 66 positives, DB3 – 22 positive), it is not very predictions of the data from models built on these limited positives is not very reliable. Hence we can artificially increase the number of positives in these data sets to see if the resulting extended data set would produce reliable results. In our experiments, we chose to double the number of positives.

5.5.4.1. Extended-set linear classifier

Here, we increase the number of positives available by modeling the kernel principal component analysis data as a Gaussian distribution and then generating random data consistent with this distribution. The random data generation falls into two steps:

- estimate mean vector and covariance matrix
- generate extra positives by using the estimated mean and covariance vectors.

We added 66 artificially generated positives to the existing 66 positives in the case of the DB2 data set and to the existing 22 positives in DB3, we added 22 artificial positives. The Linear classifier is then applied on these extended data sets which produced the following ROC curves.

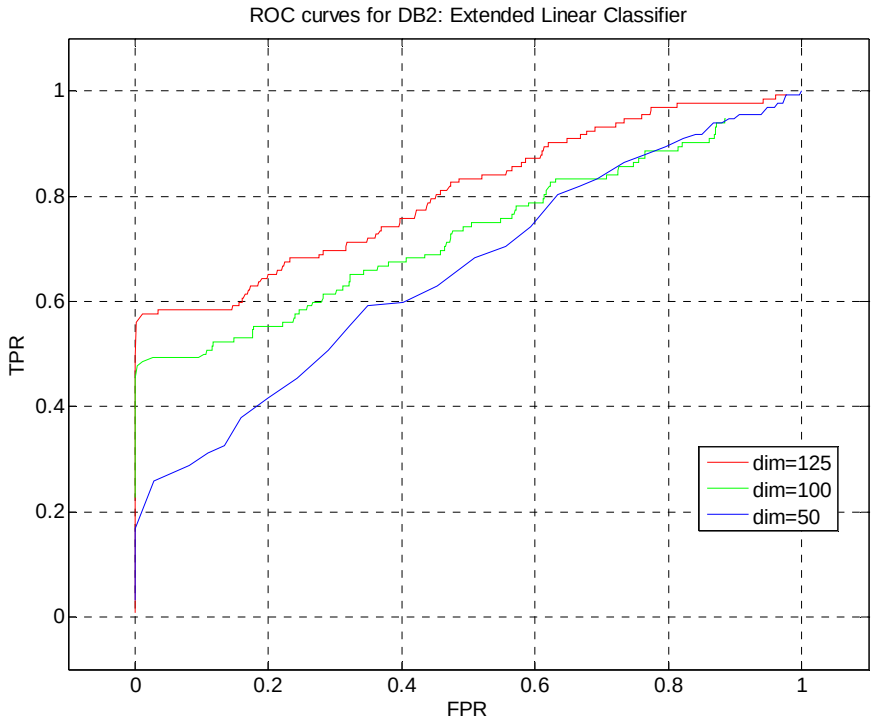

Figure 5.8 ROC curves for DB2 using Extended Linear Classifier

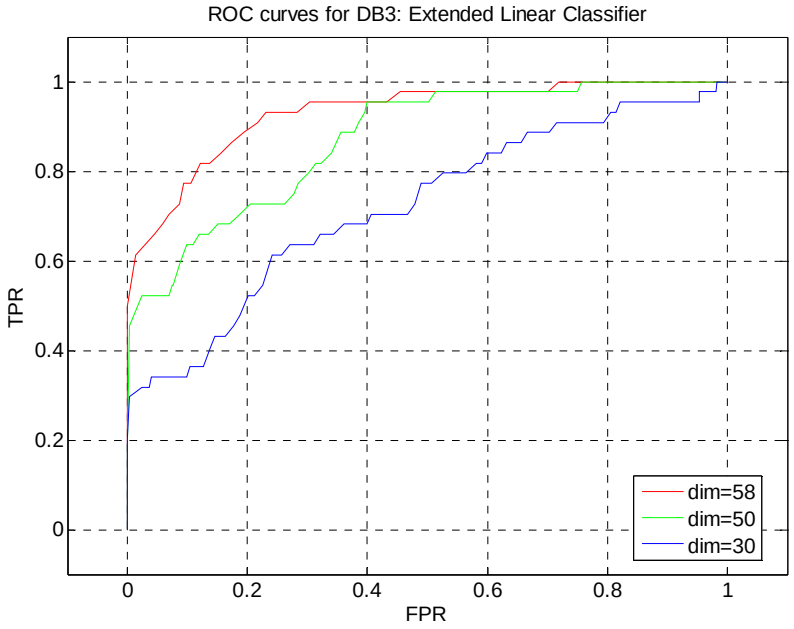

Figure 5.9 ROC curves for DB3 using Extended Linear Classifier

5.5.4.2. Extended-set elliptic classifier

The extended positives were same as in the case of Extended-set linear classifier for both DB2 and DB3. The resulting ROC curves by using Elliptic classifier on the extended data set are as follows:

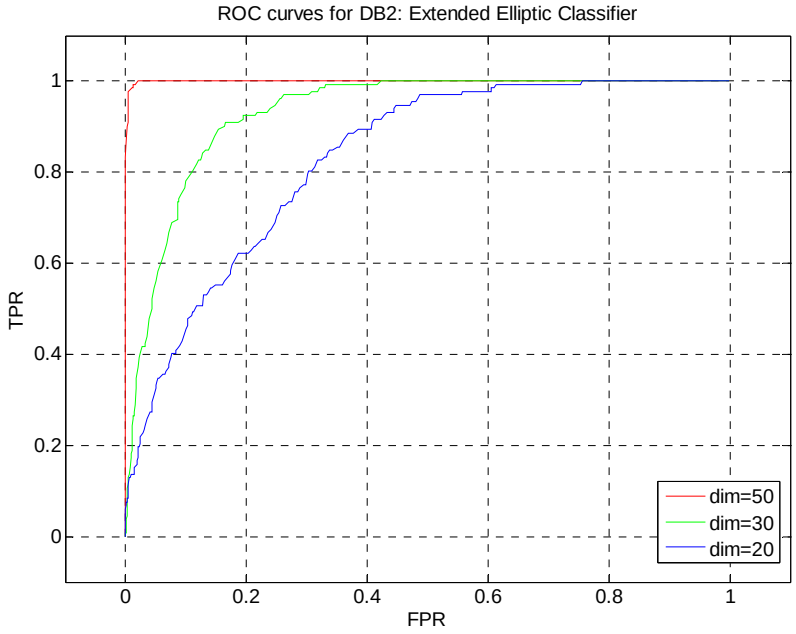

Figure 5.10 ROC curves for DB2 using Extended Elliptic Classifier

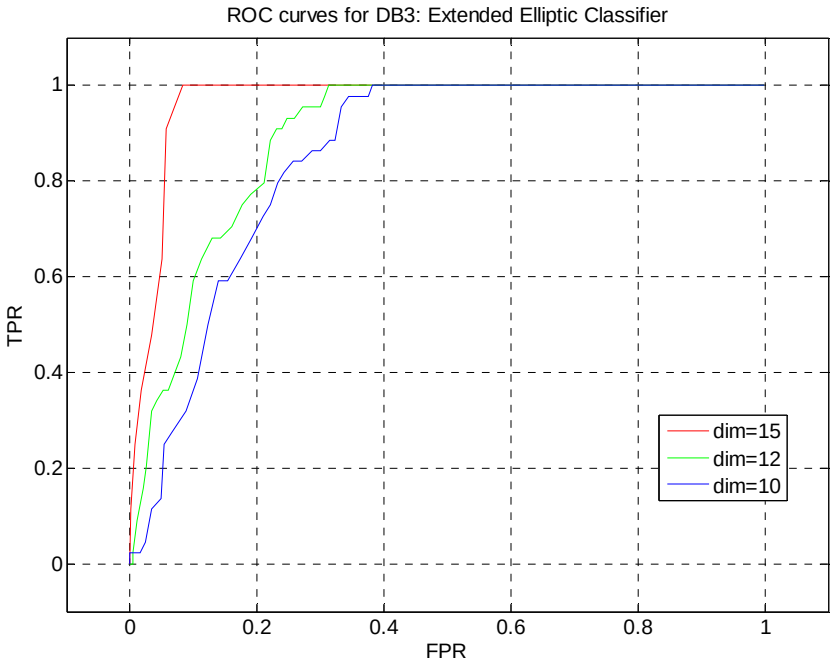

Figure 5.11 ROC curves for DB3 using Extended Elliptic Classifier

5.5.5. Comparison:

The following ROC graphs show the comparison between the performance of the linear and Elliptic classifiers on DB2 and Db3. From these two graphs it is clear that both the classifiers work better on DB3. This may be because we are dealing with just 22 positives in the case of Db3 as opposed to 66 positives in DB2.

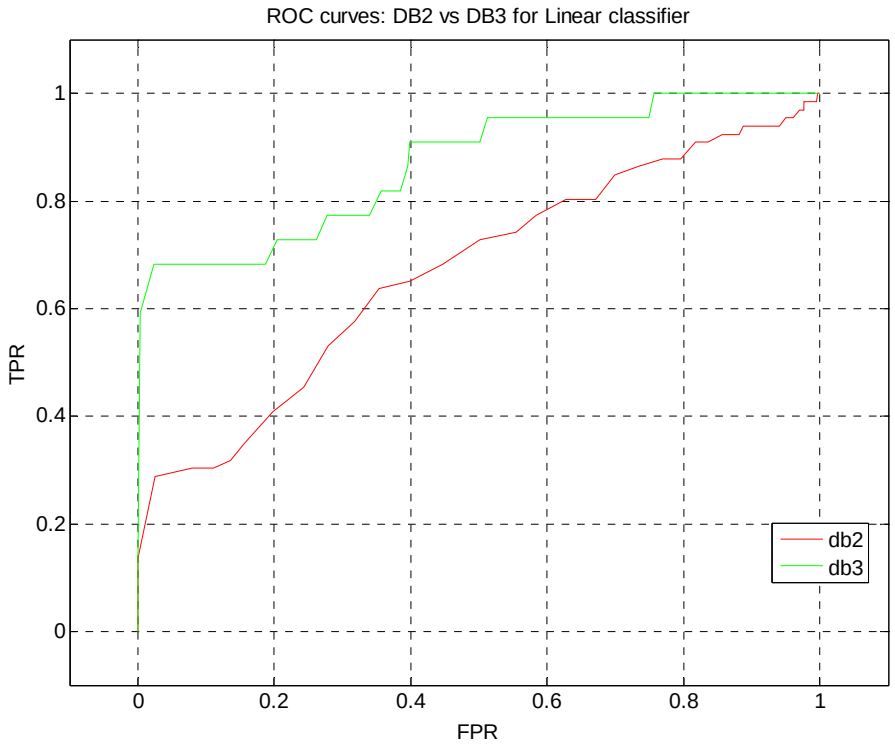

Figure 5.12 ROC curves for DB2 Vs DB3 using Linear Classifier

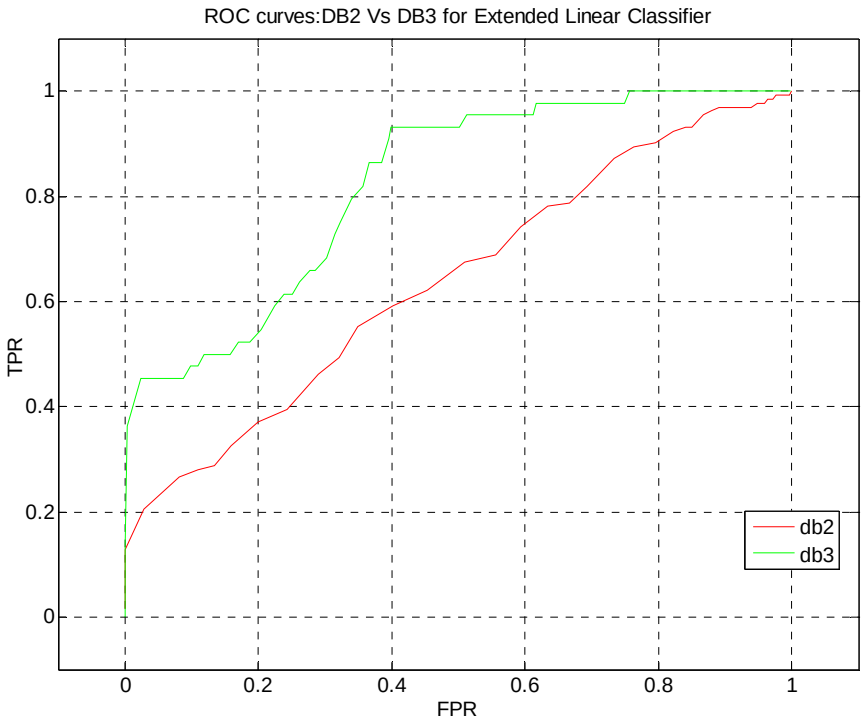

Figure 5.13 ROC curves for DB2 Vs DB3 using Extended Linear Classifier

The following graphs show the comparison of the performance of the classifiers on actual and extended data sets.

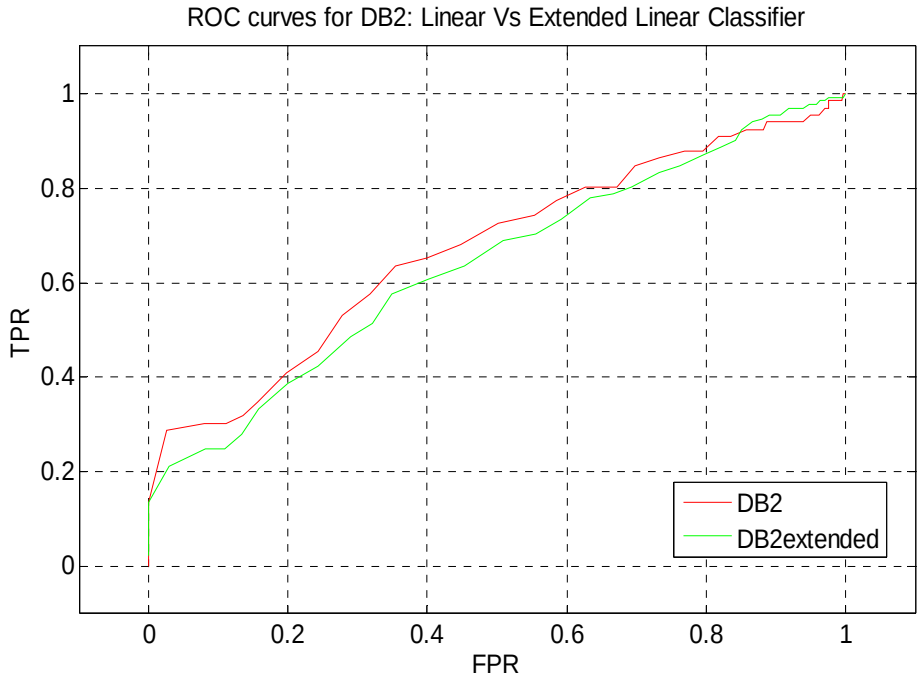

Figure 5.14 ROC curves for DB2 using Linear Vs Extended Linear Classifier

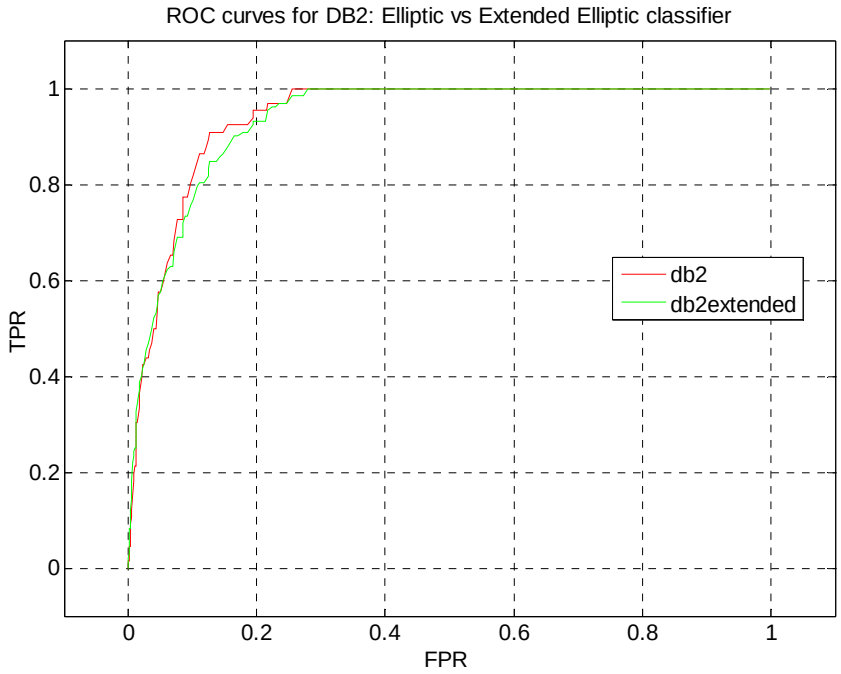

Figure 5.15 ROC curves for DB2 using Elliptic Vs Extended Elliptic Classifier

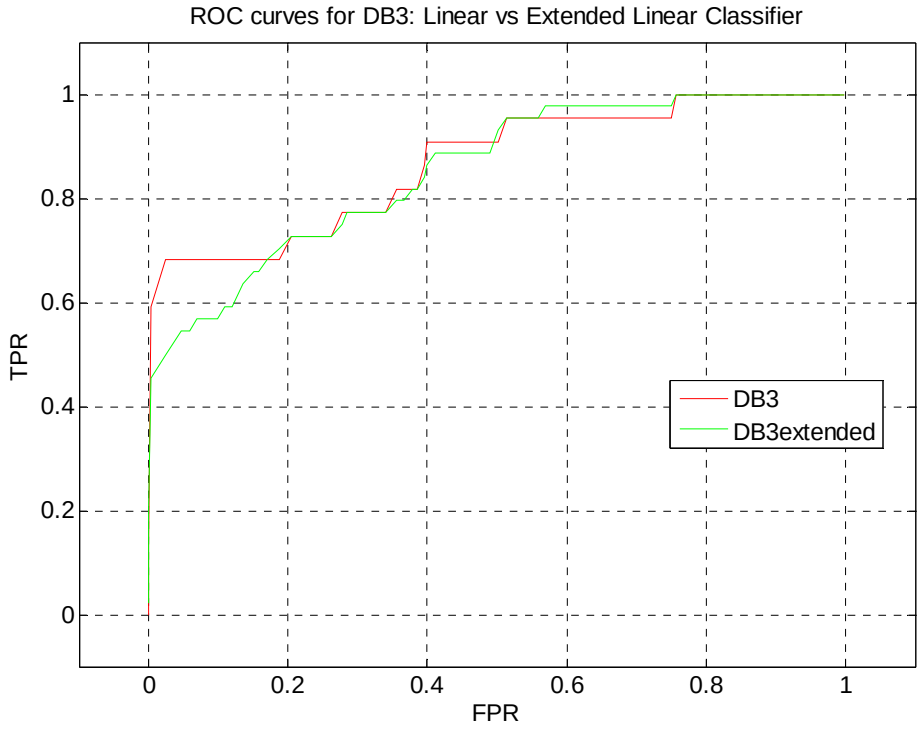

Figure 5.16 ROC curves for DB3 using Linear Vs Extended Linear Classifier

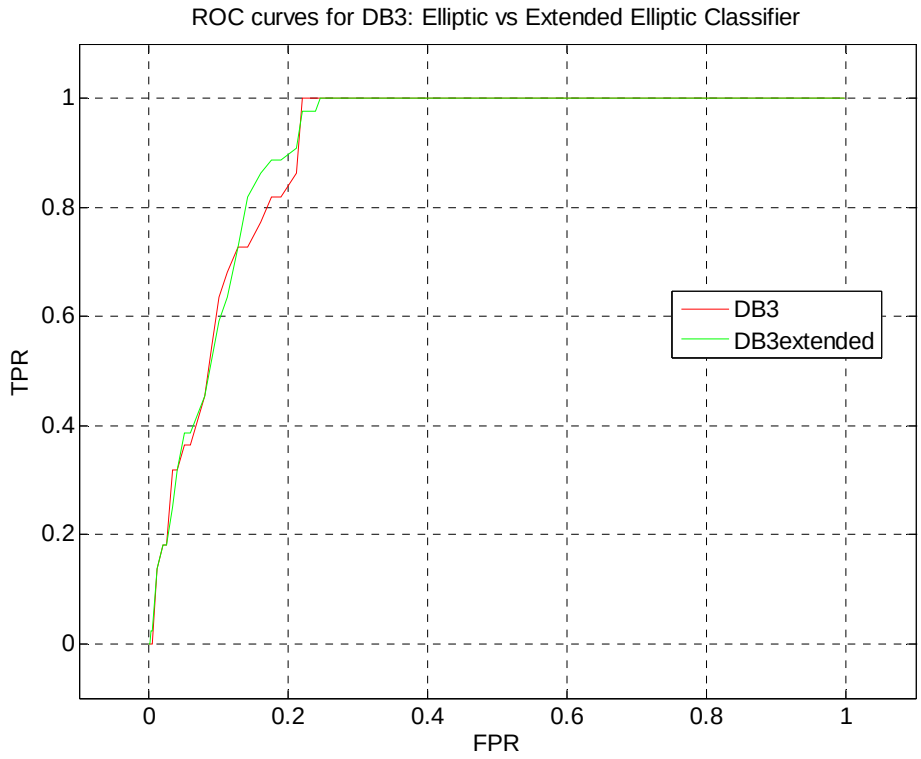

Figure 5.17 ROC curves for DB3 using Elliptic Vs Extended Elliptic Classifier

It can be seen that both in the case of DB2 and DB3, there is no significant improvement in the performance of the classifier when operating on the extended data set. This is due to small number positives in each data. Even though we are doubling the number of positives in each data set, the resulting data set with extended positives is not very reliable in estimating any new positives. For example we are dealing with 132 positives in 30 dimensions for DB2 in elliptic classifier and 44 positives in 12 dimensions for DB3. This data is not sufficient to build a very reliable model that can operate on a data set with large number of positives.

5.6. Summary

These results show that proposed Data Dependent Composite Kernel method is superior to the performance of Single kernel both in terms of Classification Accuracy as well as Mean Square Reconstruction Error. The time complexity of AKFA is $O(\ell n^2)$, which was more efficient than the complexity $O(n^3)$ of a more systematic principal component analysis (KPCA). Therefore, Accelerated Kernel Feature Analysis (AKFA that is derived

from the Sparse Kernel Feature Analysis (SKFA) is a faster and more efficient feature extraction algorithm and is an efficient means for the detection of polyps on CT colonographic images. The polyp classification experiment showed that AKFA yields the same level classification performance to that of KPCA using a *k*-nearest neighbor classifier, thus demonstrating that the features extracted by AKFA are practically useful in discrimination of polyps from false positives detections for smaller data sets. However in the case of large data sets the discrimination ability of KNN is poor because of low rate of true positives. Linear and Elliptical classifiers gave superior performance when compared to KNN classifier and the performance of linear and extended classifiers is consistent with that of these classifiers operated on Extended data sets with more number of true positives. This indicates that the available true positives in the large data sets are not sufficient to build a very reliable model. Therefore we developed methods (that are mentioned in the following chapters) to further analyze the larger data sets with few positives in them.

CHAPTER 6 ONLINE KERNEL ADAPTATION ANALYSIS

6.1 Introduction:

Traditionally, CAD schemes have been developed for off-line applications, where resources and training requirements are proportional to the number of training instances. Thus, if more training data are collected after the initial tumor model is computed, retraining of the model becomes imperative in order to incorporate the incremental data and preserve the class concept. Hereafter, we will use the term on-line learning (as opposed to batch learning) to refer to incorporating new data that become available in the already computed model.

The concept of on-line learning has been discussed as an efficient method for non-linear and on-line data analysis [37,38]. It has been applied to kernel based and non-kernel based feature extraction methods. In [38], Zheng proposed a method, similar to batch learning, where the input data was divided into a few groups of similar size, and KPCA was applied to each group. A set of eigenvectors was obtained for each group and the final set of features was obtained by applying KPCA to a subset of these eigenvectors. The application of the on-line concept to Principal Component Analysis is mostly referred to as Incremental Principal Component Analysis [39-43] and has shown computational effectiveness in many image processing applications and pattern classification systems. The effective application of on-line learning to non-linear spaces is usually undertaken by using kernel based methods [44-48].

The key idea behind the proposed method is to allow the feature space to be updated as the training proceeds with more data being fed into the algorithm. The feature space update can be incremental or non-incremental. In incremental update, the feature space is augmented with new features extracted from the new data, with a possible expansion to the feature space if necessary. In non-incremental update, the dimension of the feature

space remains constant where the newly computed features may replace some of the existing ones.

6.2 Off – Line Training.

Off-line training is the first batch of training data before the subsequent batches of new data are used for on-line learning. The eigenspace we get from the first off-line learning will include dominant information of the training data; hence, we have to be very careful during off-line training. Usually, the first batch of data contains more data than the subsequent batches. During the first step we perform off-line training, whereas subsequent batches of new data are used for on-line learning.

For the initial Off- line training, we use the method presented in Chapter 4. From the preliminary experimental results presented in Chapter 5, Sec 5.2, it is clear that the performance of sigmoid kernel is poor for all the data sets. So, from now on, in the subsequent methods presented, we avoid using sigmoid kernel. We find the Kernel Gram Matrix for the off-line data by computing the data dependant Composite kernel for offline data is computed according to Eq. (41). After computing $K_{comp}(\rho)$ for the offline data, we proceed with the training of online data as described in the subsequent sections.

6.3 Online Data Analysis

Whenever the training properties of the data change dynamically or when there is an efficient use of limited storage space, it is very essential to perform online learning. Online learning is different from stochastic or batch training. In online training, each pattern is presented once and only once. Thus, additional memory for storing the patterns is not necessary [22]. Therefore, to facilitate online training we need a method that requires retention of only a portion of the training data. The problem is in determining what this portion of data is. Hence, in our paper we present a novel way of classifying the incoming data as either heterogeneous or homogeneous and then update the feature space accordingly.

Since we have trained our system with the offline data, we extend this methodology to online stream of data. Let $\{x_i (i = 1,2,..., n)$ be n input samples, $y_i = \{+1, -1\}$ represents the class labels of *n* samples of the off-line training data, and $\{x_i^{'} (i = n+1, n+2, ..., n')\}$ be another n'-n samples that will be our new online data where the class labels of on-line training data are unknown

Our goal is to find a composite kernel that will best fit this new data. As the first step we need to extend our data dependant Kernel matrices (for 4 kernels from Eq. (20) to Eq. (23) of offline data to accommodate the information from our new online stream of data. Since the composition of matrix P_r is now changed because of the incorporation of new incoming online data, there will be a change in both q_r, α_r and hence K_r. Let us denote the new data dependant kernel matrices that have the information from both offline and online data as:

$$K_r^{'} = [q_r^{'}(x_i) q_r^{'}(x_j) p_r^{'}(x_i, x_j)]_{n' \times n'} \tag{53}$$

Where, *i,j=1,2,....n'* and *r=1,2,3,4* (*r* denotes the kernel matrices) and the vectors $q_r^{'}$,and $\alpha_r^{'}$ where we have $q_r^{'} = K_0^{'} \alpha_r^{'}$ will be $\{q_r(x_1), q_r(x_2), ... q_r(x_{n'})\}^T$ and $\{\alpha'_0, \alpha'_1, ... \alpha_{n'}\}_r^T$ respectively. The base matrix $K_0^{'}$ will now be a $n^{'} \times (n^{'}+1)$ matrix instead of a $n \times (n+1)$ matrix. As mentioned in section 3.1 Eq. (29) and Eq. (30), the Eq. (53) can now be written as:

$$K'_r = Q'_r P'_r Q_r' \tag{54}$$

Now let us find the new combination vector $\alpha_r^{'}$. Because of the addition of the new online data to the offline data, there will be a change in the class seperability of the entire data. The new class seperability factor can be given as:

$$J'(\alpha_r') = \frac{\left(\alpha'\right)_r^T M_{0r}' \alpha_r'}{\left(\alpha'\right)_r^T N_{0r}' \alpha_r'} \tag{55}$$

6.3.1 Incorporating Online data into existing system

The next step is to find out M_{0r}', N_{or}'. For finding these matrices, we need to first calculate B_{0r}', W_{or}'. The problem is, however, we need to know the class labels of the new incoming data to compute these matrices. For simplicity let us consider that we received only one online data i.e. , $x_{(n+1)}$. When this data comes in, we don't know to which class this data belongs to. But for implementing the data dependent kernel, we need to know the label of this data because we have to divide the matrix P according to the label. Therefore, we compare $x_{(n+1)}$ with mean of both class1 data and class2 data to see the closeness of the new unclassified data with that of the offline classified data. We append this new incoming online data either to the class1 or to class2 depending on the deviation from the mean of the respective classes. Let the mean of first n1 data be "m1" and the mean of remaining n2 data be "m2". They can be expresses as:

$$m_1 = \frac{1}{n_1}\sum_{i=1}^{n_1} x_i(n) \qquad and \qquad m_2 = \frac{1}{n_2}\sum_{i=n-n_1}^{n} x_i(n) \tag{56}$$

Where *n1+n2 = n*, the size of the offline data. When we get the new data, we compare it with *m1* and *m2* and append the new data to one of them. If, $m_1 - x_i\left((n+1) \le \varepsilon_1\right.$ then $x_i\left(n+1\right) \in class1$; else if $m_2 - x_i\left((n+1) \le \varepsilon_1\right.$ then $x_i\left(n+1\right) \in class2$. Once we have the label for the new online data, then we proceed to construct the new input matrices for *r*= 1,2,..,4 kernels as:

$$P_r' = \begin{pmatrix} (P')_{11}^r & (P')_{12}^r \\ (P')_{21}^r & (P')_{22}^r \end{pmatrix} \tag{57}$$

Where the sizes of the sub matrices $(P')^r_{11}, (P')^r_{12}, (P')^r_{21}, (P')^r_{22}$ are $n_1' \times n_1'$, $n_1' \times n_2'$, $n_2' \times n_1'$,, $n_2' \times n_2'$, respectively. If the new data is in class1, then $n'_1 = n_1 + 1, n'_2 = n_2$; else $n'_1 = n_1, n'_2 = n_2 + 1$. Therefore, the only extra computation will be the last row and last column of either $(P')^r_{11}, (P')^r_{12}, (P')^r_{21}$ or $(P')^r_{12}, (P')^r_{21}, (P')^r_{22}$ depending on whether the data belongs to class1 or class2 respectively. Once we compute P_r' we can easily compute the matrices $B_{or}', W_{0r}', M_{0r}', N_{or}'$ according to Eq. (34) to Eq. (47) described in Section 4.3.

To maximize $J'(\alpha'_r)$ in the Eq. (33), the standard gradient approach is followed. If matrix N'_{0r} is nonsingular, the optimal α'_r that maximizes the $J'(\alpha'_r)$ is the Eigen vector corresponding to the maximum Eigen value of the system,

$$M'_{0r}\,\alpha'_r = \lambda'_r\, N'_{0r}\,\alpha'_r \tag{58}$$

$$\lambda'^{*}_r = \arg\max_{\lambda}(N'^{-1}_{or} M'_{or}) \tag{59}$$

6.3.2 Finding the most Dominant Kernel.

Once we know the Eigen vectors, *i.e.*, combination coefficients of the composite data (both offline +online), we can compute q'_r and hence Q'_r to find out the 4 Gram Matrices corresponding to 4 different kernels by using Eq. (30). Now first *p* kernels corresponding to fist *p* Eigen values (arranged in the descending order) will be used in the construction of a composite kernel that will yield optimum classification accuracy. Before proceeding to the construction of composite kernel, we have to find whether our new data is homogeneous or heterogeneous and update our Gram Matrices accordingly. Obtaining a higher dimensional feature space of the online data with greater classification power depends on how effectively we are updating the Gram Matrix. Since construction of Gram Matrix is the crucial step for performing Feature analysis, care should be taken

not to allow any redundant data in the Gram Matrix and at the same time data of different nature should be preserved for further training of the algorithm. Hence, there is a need to distinguish the incoming data as either homogeneous or heterogeneous and process it accordingly.

If the data is homogeneous, most of the data is redundant and hence we simply discard these homogeneous data. On the contrary if the data is heterogeneous, we update the Gram Matrix either incrementally or non-incrementally depending on the level of heterogeneous nature of the online data. The following subsections will walk us through updating the Gram Matrix in accordance with the nature of the data.

6.4. Update based on nature of online data

We use Class Separability as a measure to identify whether the data is either Heterogeneous or Homogeneous. If the data is Homogeneous, then it improves the Separability and, conversely, the Heterogeneous data degrades the class separability ratio. Let us introduce a variable ξ which is the ratio of the class separability of the composite (both off-line and current online) data and the off-line data. It can be expressed as:

$$\xi = \frac{J_*^{'}(\alpha'_r)}{J_*(\alpha_r)} \tag{60}$$

where $J_*^{'}(\alpha'_r)$ denotes the class separability yielded by the most dominant kernel (which is chosen from the 4 different kernels) for the composite data (i.e. new incoming data and the previous off-line data) and $J_*(\alpha_r)$ is the class separability yielded by the most dominant kernel for off-line data. Because of the properties of class separability the Eq. (39) can be rewritten as:

$$\xi = \frac{\lambda_*^{'}}{\lambda_*} \tag{61}$$

Where $\lambda_*^{'}$ correspond to the most dominant eigenvalue of composite data (both off-line and online) and λ_* is the most dominant eigenvalue of the 4 different kernels for the off-line data. If ξ is less than a threshold value η then the incoming online data is heterogeneous else it is homogeneous.

6.4.1 Homogeneous online data

In the case of homogeneous data, we do not update the Gram Matrix. Instead we discard all the data that is homogeneous. Hence, the updated Gram Matrix (Which is exactly the same as Eq. (30) can be given as:

$$K_r^{n'} = Q_r P_r Q_r \tag{62}$$

6.4.2 Heterogeneous online data

If the new batch of incoming data is heterogeneous, the feature space has to be updated. But this update can be either incremental or non-incremental. We propose a novel way to determine if the data is highly heterogeneous or less heterogeneous and there by update the gram matrix either incrementally or non – incrementally by introducing a criterion called "Residue Factor."

We use the information from only the most dominant kernel for determining the Residue Factor. The class separability of the most dominant kernel for the new data is directly dependant on both the maximum combination coefficient $\alpha_*^{'}$ (this is the maximum combination coefficient of 4 different kernels) as well as the maximum eigen $\lambda_*^{'}$. Let us denote $\overline{\alpha_*^{'}}, \overline{\alpha_*}$ as the mean of combination coefficients $\hat{\alpha}_*^{'}, \hat{\alpha}_*$ respectively, for the most dominant kernel among the four kernels available. Using these values, Residue Factor "*rf*" is defined as:

$$rf = (\overline{\alpha_*^{'}} - \overline{\alpha_*}).\frac{\lambda_*^{'}}{\lambda_*} \tag{63}$$

This 'Residue Factor' is used to determine whether the update should be incremental or non-incremental. This is done by evaluating disparities between Eigen values of composite (online + off-line) and off-line data. This method is presented in the following subsections.

6.4.2.1. Non – incremental Update

If *rf* is less than 1, i.e. , $\|rf\| \leq 1$ then the update is non-incremental. Hence, the dimensions of the Gram matrix have to remain constant. So, we have to replace the trivial rows and columns of the previous Gram matrix with that of the new data. Since we assigned the incoming data to one of the class, we just compare combination coefficient values of that class with the combination coefficient of new incoming data to yield the difference vectors that will determine the trivial combination vector to be replaced. This process should be repeated for all the kernel matrices.

$$\Delta_{rk} = \alpha_r^{'} - \alpha_{rk} \tag{64}$$

where k = 1,2,… n_i, i=1,2, i.e. the number of classes, n_i is the number of input vectors in either class 1 or class 2 respectively and $\alpha_r^{'}$ is the combination coefficient of the $x_{(n+1)}$ just arrived data. The combination coefficient corresponding to the minimum of this difference vector, i.e. $\min(\Delta_{rk})$, will be replaced by $\alpha_r^{'}$. Let $\alpha_{r(\min\Delta)}$ be the combination coefficient corresponding to minimum difference vector, then

$$\alpha_{r(\min\Delta)} \leftarrow \alpha_r^{'} \tag{65}$$

This can be shown as follows:

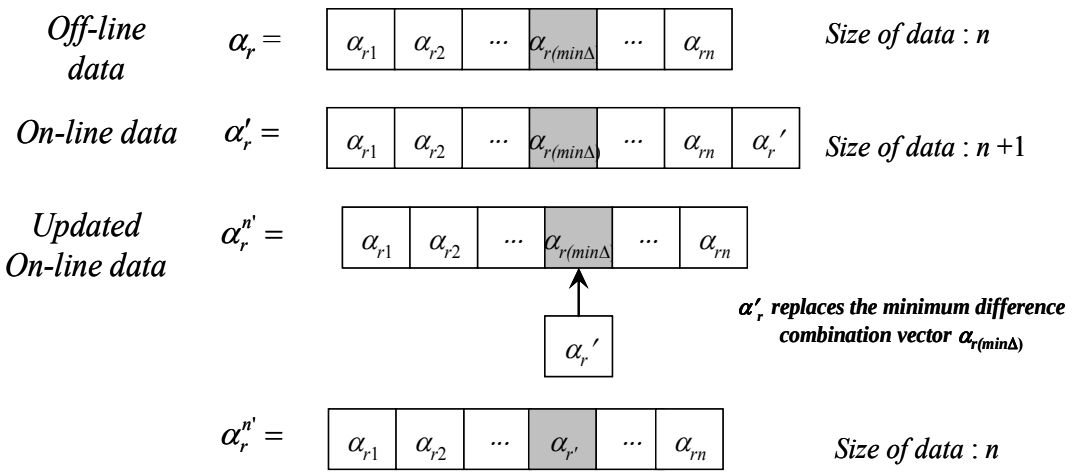

Figure 6.1 This figure shows the non-incremental update of combination vector.

Similarly the input matrix P' and Q' should also be updated by removing the row and column corresponding to the index $\alpha_{r(\min\Delta)}$ of and replacing it with the row and column corresponding to the index $\alpha_r^{'}$ .Then the new input matrix $P_r^{n'}$ can be written as:

$$P_r^{n'} = [P_r^{'}] - [P_{r(r\min\Delta,1:(n+1))}] - [P_{r(1:n,r\min\Delta)}] \tag{66}$$

This can be shown as follows:

$$P_r^{n'} = \begin{bmatrix} p_{r(1,1)} & \cdot & p_{r(1,\min\Delta)} & \cdot & p_{r(1,(n+1))} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ p_{r(\min\Delta,1)} & \cdot & p_{r(\min\Delta,\min\Delta)} & \cdot & p_{r(\min\Delta,n+1)} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ p_{r((n+1),1)} & \cdot & p_{r((n+1),\min\Delta)} & \cdot & p_{r((n+1),(n+1))} \end{bmatrix}_{(n+1)\times(n+1)} - [P_{r(\min\Delta,1:(n+1))}] - [P_{r(1:n,r\min\Delta)}]$$

$$= \begin{bmatrix} p_{r(1,1)} & \cdots & p_{r(1,\min\Delta)} & \cdots & p_{r(1,(n+1))} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{r(n,1)} & \cdots & p_{r(n,\min\Delta)} & \cdots & p_{r(n,(n+1))} \end{bmatrix}_{(n)\times(n+1)} - [P_{r(1:n,\min\Delta)}]$$

$$= \begin{bmatrix} p_{r(1,1)} & \cdots & p_{r(1,(n+1))} \\ \vdots & \ddots & \vdots \\ p_{r(n,1)} & \cdots & p_{r(n,(n+1))} \end{bmatrix}_{n \times n}$$

After computing $q_r^{n'}$, we can compute $Q_r^{n'}$ as:

$$Q_r^{n'} = diag(\alpha^{n'}) \tag{67}$$

Hence, in the non-incremental update of heterogeneous online data, the Gram Matrix can be given as:

$$K_r^{n'} = Q_r^{n'} P_r^{n'} Q_r^{n'} \tag{68}$$

6.4.2.2. Incremental Update

If the Residue Factor is greater than one, i.e. $\|rf\| \geq 1$, then that means the similarity between the previous eigenvectors the new eigenvectors is very less. So it is very important for us to retain this data which is highly heterogeneous in nature for efficient classification. So, instead of replacing the trivial rows and columns of the previous data, we simply retain it and hence the size of the Gram Matrix is incremented by the size of the new data. In this case, since we have already calculated the new combination coefficient $\alpha_r^{'}$, input matrix P' and Q' are same as in Section 4.2.1. In this case of Incremental Update, the Kernel Gram Matrix can be given as:

$$K_r^{n'} = Q_r^{'} P_r^{'} Q_r^{'} \tag{69}$$

Once we have our Kernel Gram Matrices, we can now proceed towards finding the data dependant Composite kernel gram matrix that will be expressed as linear combination of most dominant kernels for the updated system.

6.5. Data Dependant Composite Kernel for updated system of data

After computing the gram matrices of *2* most dominant kernels for the composite data (off-line + online data), we now have to combine them to yield the best classification performance. As defined in the section 4.4 we can write the composite kernel for the new composite data as:

$$K_{comp}^{n'}(\rho) = \sum_{i=1}^{p} \rho_i' Q_i^{n'} P_i^{n'} Q_i^{n'} \tag{70}$$

Where, the composite coefficient set $\hat{\rho}'$ is the collection of combination coefficients ρ_i' which are scalars and *p* is the number of kernels we intend to combine. Here, we are considering the combination of 2 kernels. We use the same alignment factor methodology to determine the optimum composite coefficients that will yield the best Composite Kernel. Let our new label vector be $y^{n'}$ and its transpose as $(y^{n'})^T$. Therefore, the new alignment factor can be written as:

$$A'(k^{n'}, y^{n'}(y^{n'})^T) = \frac{\left\langle K^{n'}, y^{n'}(y^{n'})^T \right\rangle_F}{\left\| K^{n'} \right\|_F \left\| y^{n'}(y^{n'})^T \right\|_F} = \frac{(y^{n'})^T K^{n'} y^{n'}}{n' \left\| K^{n'} \right\|_F} \tag{71}$$

We can determine the composite coefficient $\hat{\rho}'$ that will maximize the alignment factor follows:

$$\begin{aligned}
\hat{\rho}' &= \arg_{\rho'} \max \left(A' \left(\rho', k^{n'}, y^{n'}(y^{n'})^T \right) \right) \\
&= \arg_{\rho'} \max \left(\frac{\left\langle \sum_i \rho_i' K_i^{n'}, y^{n'}(y^{n'})^T \right\rangle}{n' \sqrt{\left\langle \sum_i \rho_i' K_i^{n'} \right\rangle, \left\langle \sum_j \rho_j' K_j^{n'} \right\rangle}} \right) \\
&= \arg_{\rho'} \max \left(\frac{1}{n'^2} \cdot \frac{(\rho_i')^T U^{n'} \rho_i'}{(\rho_i')^T V^{n'} \rho_i'} \right)
\end{aligned} \tag{72}$$

Where $u_i^{n'} = \left\langle K_i^{n'}, y^{n'}(y^{n'})^T \right\rangle, U_{ij}^{n'} = u_i^{n'} u_j^{n'}, V_{ij}^{n'} = v_{ij}^{n'} = \left\langle K_i^{n'}, K_j^{n'} \right\rangle$. Let the new generalized Raleigh coefficient be:

$$J(\rho') = \frac{(\rho')^T U^{n'} \rho'}{(\rho')^T V^{n'} \rho'} \tag{73}$$

We can obtain the value of $\hat{\rho}'$ by solving the generalized eigenvalue problem $U^{n'}\rho' = \delta' V^{n'}\rho'$ where δ' denotes the eigenvalues. We choose a composite coefficient vector that is associated with the maximum eigenvalue. Once we determine the best composite coefficients, we can compute the Composite kernel according to Eq. (70). After we build our model by using Composite Kernel over the entire training data and perform AKFA, we can successfully implement this training model for other online data sets as well as the test data set. In the next chapter, we see how to handle the online data over a long period of time. The following figure 7 shows the training of online data.

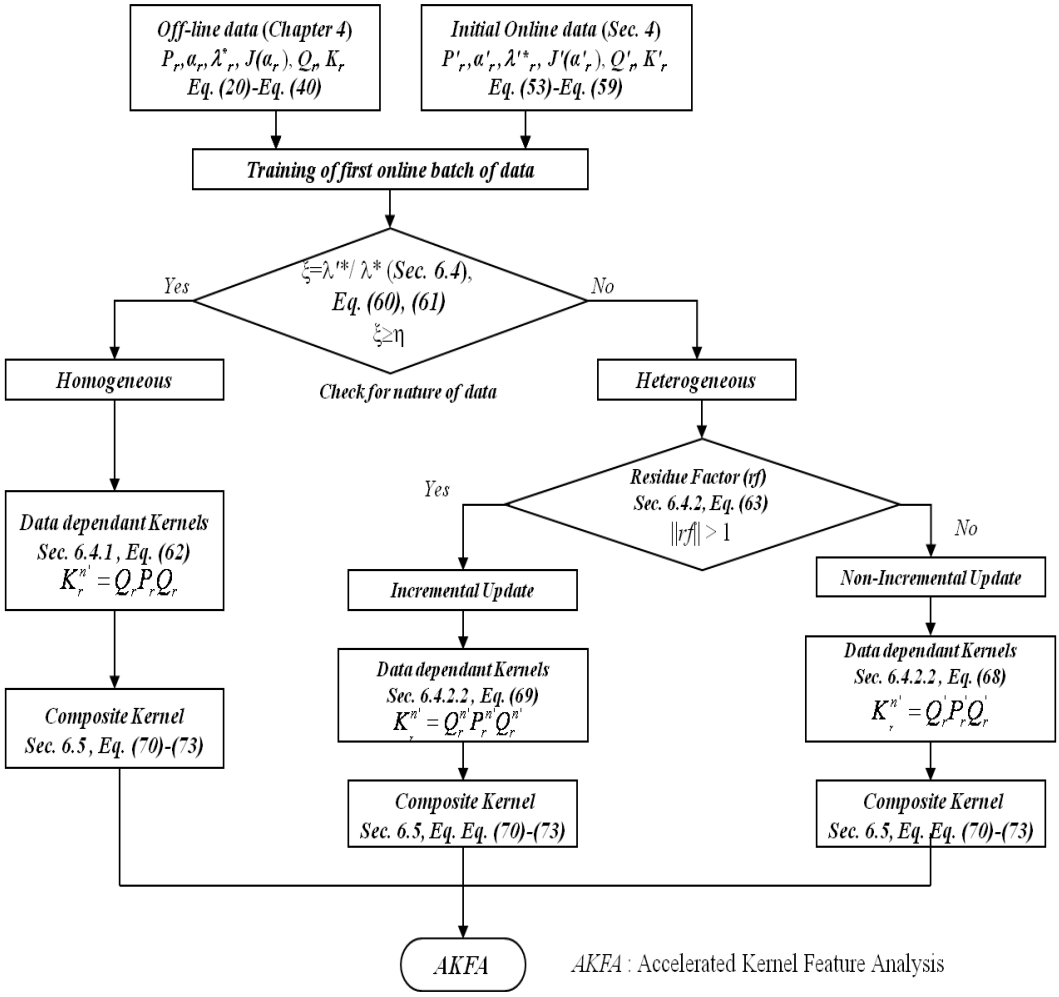

Figure 6.2 Training of First Online batch of data.

CHAPTER 7 LONG TIME SEQUENTIAL TRAJECTORIES

7.1 Introduction

In real time situations, the data acquired over a long period of time can sometimes be highly diverse, and each data set is unique in nature, obtaining a clear distinction between heterogeneous and homogeneous large online datasets is a very challenging task. Hence, there is a chance of misinterpretation of the data to be either homogeneous or heterogeneous in nature while training the new incoming online datasets. In such cases, it is often very important to reevaluate and change the criteria established in training the algorithm to correctly train the data according to its nature. In the following subsections, we explore why and how the online data should be subdivided over the time.

7.2 Reevaluation of nature of large online data

The size of the data we are considering will become a major concern when we acquire huge or time sequential data. In this case, it is not appropriate to classify the entire data as homogeneous or heterogeneous at a time, which may give erroneous results. To avoid this problem, we may find a suitable window of online data that can be taken at a time and train the algorithm according by identifying whether this new window of incoming data as either heterogeneous or homogeneous.

One way is to track the classification accuracy. If there is no compromise in the classification accuracy when we train the algorithm without further modifications (as mentioned in the previous section), then there is no need to break down the incoming batch of data into small windows or change any parameters in the previous setting of the algorithm. However, it would simply be a waste of time if we wait to change the current

model after running the entire algorithm and then check the classification result. A better way is to identify the problem in some intermediate steps and provide some feedback loop so that either the internal parameters used in the previous step or the window size of the current incoming data can be adjusted to obtain very good classification accuracy. In this subsection, we discuss about identifying and correcting the problem in an intermediate step.

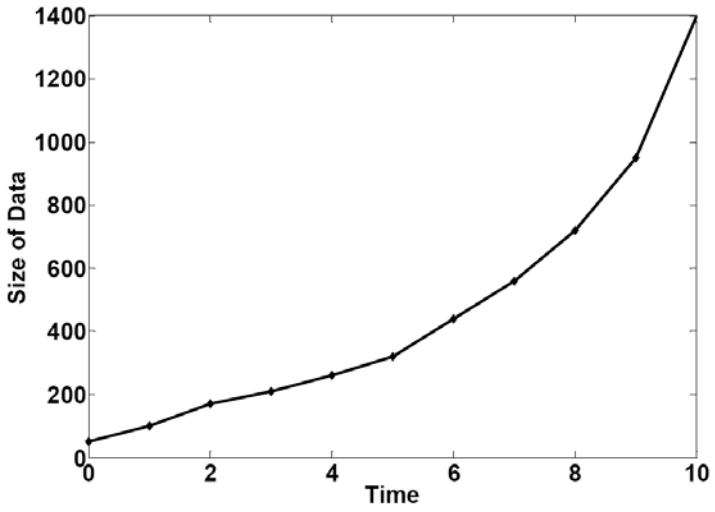

Figure 7.1 This figure shows the Relation ship between Size of the data and the time over which it acquired. As the time increases, the size of the data also increases.

For tracking whether we are correctly training the algorithm or not, we make use of Alignment factor introduced in the eq. (71) of section 6.5. The reason for using the alignment factor is that it directly influences the classification accuracy. The larger the alignment factor is, the greater the classification accuracy. Let us denote ${}^{t+1}(K_{comp}^{n'})_t$ as the update of gram matrix from time *t* to *t+1* and let ${}^{t}(K_{comp}^{n'})_0$ be the updated gram matrix from time '0' (i.e. from the beginning till one last step) to time *t*. Similarly the matrix ${}^{t+1}(y_{n'})_t$ indicates the update of the output matrix from time *t* to *t+1* and ${}^{t}(y_{n'})_0$ indicates the update of output matrix till time *t*. The alignment factor at time *t+1* and till time *t* can be given as ${}^{t+1}(A'(k_{comp}^{n'}, y_{n'}y_{n'}^{T}))_t$ and ${}^{t}(A'(k_{comp}^{n'}, y_{n'}y_{n'}^{T}))_0$ respectively. Let us introduce the criterion to track the erroneous training of the data by comparing these alignment factors

of the current data (including the new online data) and of the data till the immediate previous step as follows:

$$^{t+1}(A'(k_{comp}^{n'}, y_{n'} y_{n'}^{T}))_t < {}^{t}(A'(k_{comp}^{n'}, y_{n'} y_{n'}^{T}))_0 \quad (74)$$

This indicates that if the alignment factor deteriorates then the parameter settings in the eq. (60) should be set to a different value and the experiment should be run again. A significant variation in the alignment factor shows that the incoming online data is too large for correctly training it as either heterogeneous or homogeneous. In this case, we have to divide and keep dividing the data into several small chunks of data until we find a suitable window size of online data that would be appropriate for training and yield best results. This is discussed in the following subsection.

7.3 Decomposing online data into small chunks

The online data can be extracted into smaller size of data and consider each chunk of data at a time. We may treat small datasets as heterogeneous or homogenous data by checking the class Separability of each dataset. Initially for illustration purposes, let us consider only 10% data of the new online batch of data to find the appropriate window size for the incoming data and determine whether it is homogeneous or heterogeneous data using eq. (60). The following figure 7.2 illustrates the segmentation of the incoming online data into 10 small equal subsets. Since we divide the new online data into 10 equal sized datasets let us call them as d1, d2, d3,….d10, where the size of each new sub dataset is equal to *l*. All the new sets of data have to process sequentially but not simultaneously because of two reasons.

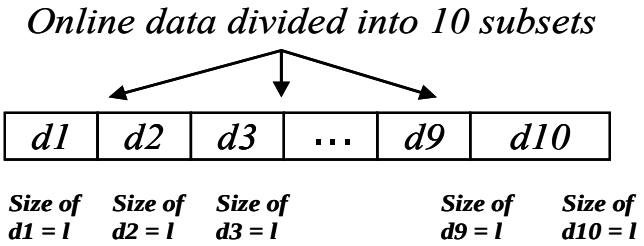

Figure 7.2. This figure illustrates the division of a large online dataset into several (10) small subsets of equal size *l*.

The first reason is that there may be a repetition of data between these small chunks of datasets. For example, d7 data may be almost similar to the d9 chunk of data. In this case, Training d7 alone will suffice. Therefore, by training the data sequentially we store the information present in the d7 chunk of data and simply discard the d9 data since all of its data is redundant.

The second reason is the size issue. If we process the data simultaneously, then each independent chunk of data would be of a different nature. Let us consider an example of this situation wherein d4 data is homogeneous, d5 is highly heterogeneous, and d6 is less heterogeneous. So, the resulting Composite Gram matrices will be of different sizes and, hence, there will be a problem in embedding all these datasets together after processing them individually. Therefore, the training of these small chunks of data has to be sequential but not simultaneous.

Now, since the training of small online datasets has to be sequential, let us consider the d1 data first. Compute the classes of the elements of the d1 data according to the equations (56). Then the input matrix at the current state, i.e. at time $t+1$ can be given as:

$$^{t+1}(P_r^{d1})_t = [\,^t(P_r^{n'})_0 \quad P_{d1}] \tag{75}$$

Similarly at time $t+1$, according to Eq. (34) to Eq. (37)

calculate: ${}^{(t+1)}(B_{0r}^{d1})_t, {}^{(t+1)}(W_{0r}^{d1})_t, {}^{(t+1)}(M_{0r}^{d1})_t, {}^{(t+1)}(N_{0r}^{d1})_t$. Using Eq. (58) and Eq. (59) calculate the class separability of the composite data (data till time *t* + d1 data) as:

$$ {}^{(t+1)}(J^{d1}(\alpha_r))_t = \frac{{}^{(t+1)}(\alpha_r^{d1})_t^T * {}^{(t+1)}(M_{0r}^{d1})_t * {}^{(t+1)}(\alpha_r^{d1})_t}{{}^{(t+1)}(\alpha_r^{d1})_t^T * {}^{(t+1)}(N_{0r}^{d1})_t * {}^{(t+1)}(\alpha_r^{d1})_t} \tag{76} $$

Therefore,

$$ {}^{(t+1)}(\xi)_t = \frac{{}^{(t+1)}(\lambda_*^{'})_t}{{}^{t}(\lambda_*)_0} \tag{77} $$

where ${}^{(t+1)}(\lambda_*^{'})_t$ is the largest eigenvalue of the data (of the dominant kernel) received from time *t* to *t+1*, ${}^{t}(\lambda_*^{'})_0$ is the largest eigenvalue of the dominant kernel calculated from time 0 to time *t*, i.e. till the previous step. Now we test whether this sub dataset d1 is of either heterogeneous or homogeneous in nature. If ${}^{(t+1)}(\xi)_t$ is greater than a new threshold value $\eta^{'}$ then the incoming data is heterogeneous; otherwise, it is homogeneous.

7.3.1 No - Update

If the data is homogeneous, then we do not update any information from the new subset d1 on to the previous gram matrix. Hence, the kernel gram matrix at time *t+1* is same as the previous one at time t and can be given as:

$$ {}^{t+1}(K_{comp}^{n'}(\rho))_t = [{}^{t}(K_{comp}^{n'}(\rho))_0]_{({}^{t}n_0)\times({}^{t}n_0)} \tag{78} $$

where ${}^{(t+1)}\left(K^{n'}_{comp}(\rho)\right)_t$ is the gram matrix that has to be completed in the current step is ; ${}^{t}(K^{n'}_{comp}(\rho))_0$ is the gram matrix computed in the previous step and $({}^{t}n_0)$ is the size of the gram matrix in the previous step.

7.3.2 Non – Incremental Update

If the data d1 is heterogeneous, then we have to decide whether it is highly or less heterogeneous by making use of the Residue Factor. In this situation, the residue factor can be given as:

$$ {}^{(t+1)}(rf^{d1})_t = \left(\overline{{}^{(t+1)}(\alpha_*^{d1})_t} - \overline{{}^{t}(\alpha_*)_0} \right) . \frac{{}^{(t+1)}(\lambda_*^{d1})_t}{{}^{t}(\lambda_*)_0} \tag{79} $$

where ${}^{t+1}(\lambda_*^{d1})_t, \overline{{}^{t+1}(\alpha_*^{d1})_t}$ are the largest eigenvalue and the corresponding mean of combination coefficient of the most dominant kernel for the composite (data till time *t* and d1), ${}^{t}(\lambda_*^{d1})_0, \overline{{}^{t}(\alpha_*^{d1})_0}$ are the eigenvalues and mean of combination coefficients from the previous step. If ${}^{t+1}(rf^{d1})_t$ is less than one then the update of the d1 data is non-incremental. When the update is non-incremental, we have to find out and replace the trivial rows and columns in the dataset from the previous step and update them with the corresponding rows and columns form the new dataset d1. Computing the trivial rows and columns as:

$$ {}^{t+1}(\Delta_{rk}^{d1})_t = {}^{t+1}(\alpha_r{}^{d1(l)})_t - {}^{t}(\alpha_{rk})_0 \tag{80} $$

where k = 1,2,… $({}^{t}n_0)$, *i=1,2*, i.e. the number of classes, $({}^{t}n_0)$ is the number of input vectors in either class 1 or class 2 respectively in the previous step, *l* the size of d1 data and $\alpha_r{}^{d1(l)}$ are the combination coefficient for the 4 different kernels of the d1 dataset. The combination coefficients in the previous step corresponding to the minimum of this

difference are replaced by the corresponding vectors in the present step. This can be represented as follows:

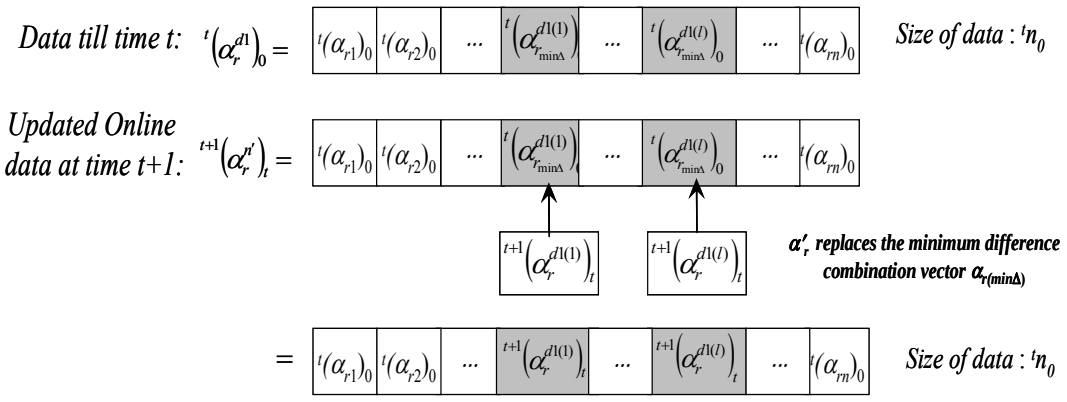

Figure 7.3 Non-incremental Update of Combination vector at time *t+1*

Similarly update ${}^{t+1}(P_r^{n'})_t$ and ${}^{t+1}(Q_r^{n'})_t$ and compute the new gram matrix as follows:

$$({}^{t+1})(K_r^{n'})_t = {}^{t+1}(Q_r^{n'})_t * {}^{t+1}(P_r^{n'})_t * {}^{t+1}(Q_r^{n'})_t \tag{81}$$

Now after determining the data-dependant kernels for d1 data, we have to determine the composite kernel using Eq. (72) and Eq. (73) as:

$${}^{t+1}(K_{comp}^{n'}(\rho))_t = [\sum_{i=1}^{p} \rho_i ({}^{t+1}(K_r^{n'})_t)]_{({}^{t}n_0)\times({}^{t}n_0)} \tag{82}$$

7.3.3 Incremental Update

If the data is highly heterogeneous, then we go for an update of the gram matrix which is incremental. In the incremental updating, we do not discard any data instead we append the data so as to preserve the diverse information that may be used to classify the next incoming chunk of online data, i.e. d2 as either heterogeneous or homogeneous in nature. Since we are using the both current and previous information, here in the case of

incremental updating, the combination coefficient will have the size of $(({}^{t}n_0) + l)$ where l is the size of d1 chunk of data. In the case of incremental updating we have:

$$\begin{aligned} {}^{t+1}(\alpha_r^{n'})_t &= {}^{t+1}(\alpha_r^{d1})_t \\ {}^{t+1}(Q_r^{n'})_t &= diag({}^{t+1}(\alpha_r^{n'})_t) = {}^{t+1}(Q_r^{d1})_t \\ {}^{t+1}(P_r^{n'})_t &= {}^{t+1}(P_r^{d1})_t \end{aligned} \tag{83}$$

With the knowledge of above matrices, the data-dependant kernel for highly heterogeneous d1 data can now be given as:

$${}^{(t+1)}(K_r^{n'})_t = {}^{t+1}(Q_r^{d1})_t * {}^{t+1}(P_r^{d1})_t * {}^{t+1}(Q_r^{d1})_t \tag{84}$$

Once we have the 4 data-dependant kernels for the new composite data at time $t+1$, we now find a composite kernel that will yield us optimum classification accuracy as:

$${}^{t+1}(K_{comp}^{n'}(\rho))_t = [\sum_{i=1}^{p} \rho_i * {}^{t+1}(K_r^{n'})_t]_{(({}^{t}n_0)+l)\times(({}^{t}n_0)+l)} \tag{85}$$

We perform the entire algorithm to see if there is an improvement in the difference of the Alignment Factors from the current and previous step. If the deviation is still huge, then the step size has to be further reduced and the algorithm should be computed again. This process should be repeated until we find an appropriate window size of d1 data that would allow us for the proper classification of the homogeneous and heterogeneous data and result in error free training.

Once we find that appropriate window size by using the d1 data, we preserve the size of this window and apply it to the rest of the datasets d2, d3, and so on until we find a significant improvement in the alignment factor in the subsequent chunks of data. If this window size fails for d2 chunk of data, then it should be further reduced for best fitting the nature of the d2 data and this should be carried forward to train the d3 data whose size is

equal to the window size determined while processing the d2 chunk of data. This procedure will be repeated for online training of any forth coming new datasets.

After the training of the gram matrix is finished to incorporate the dynamic features of the new online data, the AKFA algorithm is applied of the kernel gram matrix followed by k-nn classifier to classify the test datasets. This process is summarized in the following flow chart.

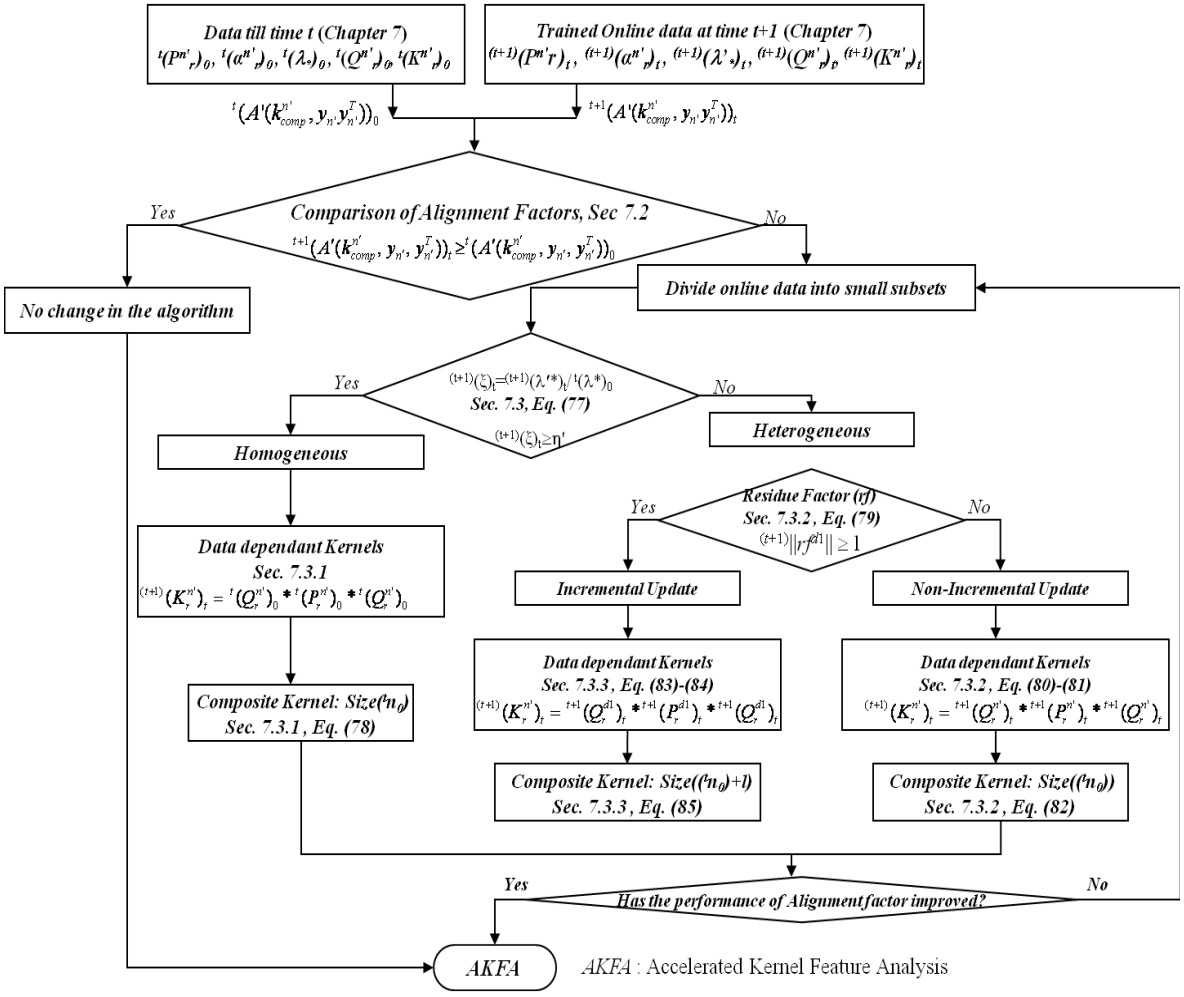

Figure 7.4. The training of online datasets acquired over longtime sequences.

Chapter 8 Experimental Results

8.1. Off-line data

The proposed Online and Offline methodologies together with AKFA are evaluated using CT image datasets of colonic polyps comprised of Positives (P) and Negatives (N) detected by our CAD system [5]. The experimental analysis is carried out in three stages. In the first step, we analyze the offline training of 4 colon cancer data sets followed by online training results in the second step and finally in the third stage we analyze the proposed method for the large data sets. The following tables show the arrangement of offline training and test sets for the 4 Colon Cancer Datasets. Instead of using cross-validation, we randomly divide the entire data into training and testing. We use the same test set for each respective Dataset through out the entire experiments.

Table 8.1
Arrangement of Off-line Datasets

Data Sets	No. of Vectors in Training Set		Total	No. of Vectors in Test Set		Total
	P	N		P	N	
Data Set1	21	69	90	8	32	40
Data Set2	38	360	398	16	300	316
Data Set3	10	500	510	6	425	431
Data Set4	7	1050	1057	4	1200	1204

8.2. Training of Off-line data

8.2.1. Dominant Kernels and their Linear Combination for Offline data sets

We used the method proposed in section 4.1 to create 4 different data-dependant kernels and select the kernel that would best fit the data and give optimum classification accuracy for the offline data. We determine the optimum kernel depending on the eigenvalue that will produce maximum separability. The following table indicates the eigenvalues λ and parameters of 4 kernels (RBF, Laplace, Linear, Polynomial) for each dataset calculated using Eq. (39).

Table 8.2
Eigen Values of 4 Kernels for Off-line Datasets

Datasets	Parameters & λ for Linear Kernel (Eq. 1)	Parameters & λ for Polynomial Kernel (Eq. 2)	Parameters & λ for RBF Kernel (Eq. 3)	Parameters & λ for Laplace Kernel (Eq. 4)
Dataset1	*λ= 10.66*	*λ= 10.19* *d =1, Offset=2*	*λ=* ***14.13*** *σ =4.12*	*λ= 12.55* *σ =0.2*
Dataset2	*λ= 123.31*	*λ= 98.79* *d =1, Offset=4*	*λ=****170.75*** *σ =5.65*	*λ= 80.57* *σ =3.5*
Dataset3	*λ=57.65*	*λ=54.03* *d =1, Offset=1*	*λ=****74.55*** *σ =5.65*	*λ=30.23* *σ =1.0*
Dataset4	*λ=72.41*	*λ=86.33* *d =1, Offset=2*	*λ=****124.13*** *σ =4*	*λ=56.35* *σ =2.5*

The kernel with the maximum value is highlighted for each offline dataset in the table2. Once we find out the kernel that yields the optimum eigenvalue, we then select the two largest kernels to form the composite Kernel. For example, Dataset1, we will combine RBF and Laplace to form the composite kernel. We have observed that each database has different combinations for the composite kernels. The Composite Coefficients of the two most dominant Kernels are listed in Table 3.

Table 8.3
The value of $\hat{\rho}$ for each of the Composite Kernels

Datasets	Two most Dominant Kernels	Linear Combination of kernels.
Dataset 1	*RBF and Laplace*	ρ_1 *=0.98,* ρ_2 *=0.14*
Dataset2	*RBF and Linear*	ρ_1 *=0.56,* ρ_2 *=0.18*
Dataset 3	*RBF and Linear*	ρ_1 *=0.98,*ρ_2 *=0.23*
Dataset 4	*RBF and Polynomial*	ρ_1 *=0.98,*ρ_2 *=0.23*

Table 3 shows the two kernel functions are combined according to the Composite Coefficients listed in the table. These composite Coefficients are obtained using the Eq. 51, listed in Section 4.4. For all the Datasets, the most dominant Kernel is the RBF kernel where as the Second most Dominant Kernel kept varying. As a result the contribution of the RBF kernel is higher when compared to other kernels in forming a Composite kernel.

8.2.2 The Classification accuracy and Mean Square reconstruction for Offline data sets

Error for the offline datasets are documented in the following table. The Mean Square Reconstruction error in the case of Offline data using AKFA algorithm is calculated as $Err_i = \left\|\Phi_i - \Phi_i^{'}\right\|^2$. The Classification accuracy is calculated as (TP+TN)/(TP+TN+FN+FP) where TP stands for "True Positive", TN stands for "True Negative," FN stands for Forced Negative" and FP stands for "Forced Positive." These results are summarized in the following table.

Table 8.4
Classification Accuracy and Mean Square Error of Off-line Data

Datasets	Mean Square Reconstruction Error of offline data with Composite Kernel (%)	Classification accuracy of offline data with Composite Kernel (%)
Dataset1	0.32	90
Dataset2	9.64	94.94
Dataset3	6.25	98.61
Dataset4	14.03	99.67

From Table 4, it is clear that we have obtained very good classification accuracy by using Composite Data-Dependant Kernel for all the offline datasets. In the next sections we documented the results of this method extended to online data.

8.3. Online Data

We follow the method mentioned in Sec. 6 to tune the selection of appropriate kernels when the new online data becomes available. The following table shows the size of different online batches of data for each colon cancer dataset.

Table 8.5
Size of Online data at different time sequences

Data Sets	Online Data Sequence #1	Online Data Sequence #2	Online Data Sequence #3	Online Data Sequence #4
Data Set1	3 P, 12N	3 P, 12N	3 P, 12N	3 P, 12N
Data Set2	7 P, 85N	7 P, 85N	7 P, 85N	7 P, 85N
Data Set3	2 P, 87 N	2 P, 87 N	2 P, 87 N	2 P, 87 N
Data Set4	2 P,126N	2 P,126N	2 P,126N	2 P,126N

8.3.1. Training of Online Data

Selection of Optimum Kernel and Composite Kernel for the new Online data. After we tentatively form the input matrices for 4 different kernels, we use equations (58) and (59) to find the dominant kernels for the new data and the previous off-line data. These results are summarized below.

Table 8.6
Eigenvalues of 4 different Kernels for Online Data

Data Sets	Online Data Sequence #1	Online Data Sequence #2	Online Data Sequence #3	Online Data Sequence #4
Data Set1	Linear $\lambda=9.99^{**}$	Linear $\lambda=9.89$	Linear $\lambda=10.16$	Linear $\lambda=9.73$
	Poly $\lambda=9.31^{**}$	Poly $\lambda=8.96$	Poly $\lambda=9.52$	Poly $\lambda=10.31$
	RBF $\lambda=\mathbf{11.34}^{**}$	RBF $\lambda=\mathbf{12.47}$	RBF $\lambda=\mathbf{18.09}$	RBF $\lambda=\mathbf{24.69}$
	Laplace $\lambda=8.41^{**}$	Laplace $\lambda=6.7$	Laplace $\lambda=12.85$	Laplace $\lambda=13.27$
Data Set2	Linear $\lambda=129.44$	Linear $\lambda=84.97^{**}$	Linear $\lambda=95.63^{**}$	Linear $\lambda=109.02$
	Poly $\lambda=106.44$	Poly $\lambda=85.01^{**}$	Poly $\lambda=96.27^{**}$	Poly $\lambda=109.61$
	RBF $\lambda=\mathbf{134.59}$	RBF $\lambda=\mathbf{101.21}^{**}$	RBF $\lambda=\mathbf{122.81}^{**}$	RBF $\lambda=\mathbf{133.40}$
	Laplace $\lambda=90.76$	Laplace $\lambda=38.49^{**}$	Laplace $\lambda=84.12^{**}$	Laplace $\lambda=115.78$
Data Set3	Linear $\lambda=75.13$	Linear $\lambda=73.88$	Linear $\lambda=62.59^{**}$	Linear $\lambda=72.45$
	Poly $\lambda=36.38$	Poly $\lambda=69.25$	Poly $\lambda=66.48^{**}$	Poly $\lambda=83.39$
	RBF $\lambda=\mathbf{80.16}$	RBF $\lambda=\mathbf{86.41}$	RBF $\lambda=73.66^{**}$	RBF $\lambda=\mathbf{109.29}$
	Laplace $\lambda=44.66$	Laplace $\lambda=13.87$	Laplace $\lambda=49.46^{**}$	Laplace $\lambda=50.25$
Data Set4	Linear $\lambda=82.72$	Linear $\lambda=52.99^{**}$	Linear $\lambda=69.43$	Linear $\lambda=98.64$
	Poly $\lambda=92.33$	Poly $\lambda=75.83^{**}$	Poly $\lambda=79.85$	Poly $\lambda=112.39$
	RBF $\lambda=132.47$	RBF $\lambda=103.25^{**}$	RBF $\lambda=\mathbf{144.59}$	RBF $\lambda=\mathbf{157.27}$
	Laplace $\lambda=38.67$	Laplace $\lambda=43.26^{**}$	Laplace $\lambda=32.59$	Laplace $\lambda=62.85$

"**" Indicates the eigen values after updating the matrix according to the method presented in Sec 6.4.

8.3.2. Classification of Heterogeneous versus Homogeneous Data

After determining the dominant kernels, the next step is to update these data-dependant kernel matrices for the computation of the composite kernel matrix which is the key step

in our experiment. We imported the criterion Eq. (61) in Sec. 6.4 to classify this new data as into the following 2 categories namely “Homogeneous” and “Heterogeneous.”

Table 8.7
Nature of different Online Datasets

Datasets	Online data Sequence #1	Online data Sequence #2	Online data Sequence #3	Online data Sequence #4
Dataset1	**Heterogeneous**	Homogeneous	Homogeneous	Homogeneous
Dataset2	Homogeneous	**Heterogeneous**	**Heterogeneous**	Homogeneous
Dataset3	Homogeneous	Homogeneous	**Heterogeneous**	Homogeneous
Dataset4	Homogeneous	**Heterogeneous**	Homogeneous	Homogeneous

The new online sequences of data for each data set analyzed by our proposed method and correspondingly, their feature spaces are updated. In the case of homogeneous data, we discard the new patterns and in the case of heterogeneous data we determine whether to non-incrementally or incrementally update the feature space. Depending on the nature of the data we update the data-dependant kernels as “No update,” “Non- Incremental Update,” “Incremental Update” according to the method Eq. (63) presented in subsections of Sec. 6.4. The results for all the online datasets are given in the following Table 8.

Table 8.8
Update of different sets of Online Data

Data Sets	Online Data Sequence #1	Online Data Sequence #2	Online Data Sequence #3	Online Data Sequence #4
Data set1	Update: Non-incremental; Gram Matrix Size:85x85	Update: No-update Gram Matrix Size:85x85	Update: No-update; Gram Matrix Size:85x85	Update: No-update; Gram Matrix Size:85x85
Data set2	Update: No-update Gram Matrix Size:398x398	Update: Non-incremental; Gram Matrix Size: 398x398	Update: Non-incremental; Gram Matrix Size: 398x398	Update: No-update; Gram Matrix Size: 398x398
Data set3	Update: No-update; Gram Matrix Size:510x510	Update: No-update; Gram Matrix Size: 510x510	Update: Non-incremental; Gram Matrix Size: 510x510	Update: No-update; Gram Matrix Size: 510x510
Data set4	Update: No-update; Gram Matrix Size:1057x1057	Update: Non-incremental; Gram Matrix Size: 1057x1057	Update: No-update; Gram Matrix Size: 1057x1057	Update: No-update; Gram Matrix Size:1057x1057

In all the data sets, we had gram matrices with either No – update or Non – incremental update. Because of the nature of the data sets, we did not have any incremental updates of the gram matrices.

8.3.3. Construction of Data Dependant Composite Kernel Matrices for Online Datasets

Once we have the data-dependant Kernel Gram Matrices for 4 different Kernels, we can now proceed to calculate the Data-Dependant Composite kernels for each of the Cancer Dataset for the new online data according to the method presented in Sec 6.4. These results are shown in Table 8.9.

Table 8.9
The value of $\hat{\rho}$ for each of the Composite Kernels of Online Data

Datasets	Online data Sequence #1	Online data Sequence #2	Online data Sequence #3	Online data Sequence #4
Dataset1	RBF &Linear $\rho_1 = 0.99$, $\rho_2 = 0.17$	*'sab'*	*'sab'*	*'sab'*
Dataset2	*'saof'*	RBF&Linear $\rho_1 = 0.93$, $\rho_2 = 0.16$	RBF&Linear $\rho_1 = 0.99$, $\rho_2 = 0.14$	*'sab'*
Dataset3	*'saof'*	*'sab'*	RBF&Polynomial $\rho_1 = 0.73$, $\rho_2 = 0.32$	*'sab'*
Dataset4	*'saof'*	RBF&Polynomial $\rho_1 = 0.90$, $\rho_2 = 0.28$	--	*'sab'*

Where, *'sab'* denotes "same as before" and *'saof'* denotes "same as offline training".

8.3.4. Classification Accuracy and Mean Square reconstruction error for Online Data

After training the online data for all the 4 cancer data sets, we evaluated the performance of our proposed model by the same test set that is used for the offline learning. The results of the classification accuracy and MSE (calculated as mentioned in Sec. 8.2.2.) are summarized in the following table 8,10 for the online data sets.

Table 8.10
Classification Accuracy and Mean Square Error for Online Datasets

Datasets	Online data Sequence #1	Online data Sequence #2	Online data Sequence #3	Online data Sequence #4
Dataset1	MSE: 0.43 Accuracy:92.5	*'sab'*	*'sab'*	*'sab'*
Dataset2	*'saof'*	MSE: 15.06 Accuracy:93.99	MSE: 14.50 Accuracy:94.94	*'sab'*
Dataset3	*'saof'*	*'sab'*	MSE: 7.68 Accuracy:96.52	*'sab'*
Dataset4	*'saof'*	MSE: 15.63 Accuracy:97.65	*'sab'*	*'sab'*

Where, *'sab'* denotes "same as before" and *'saof'* denotes "same as offline training". The classification accuracy of the online data from the above table show that these results are very close to the classification accuracy obtained during the offline training. This demonstrates that our proposed dynamic algorithm is highly efficient in handling online data while producing the results in a very short time.

8.4. Evaluation of Long Time Sequential Trajectories

Let us consider the new online data which is much larger than the previous online datasets. In this subsection, we evaluate the performance of this new large online dataset. As an initial step we perform the online learning till Chapter 6 and find out the Alignment Factor in Eq. (71) for this current Online Data Sequence (i.e., 5'th online data Sequence) of data and compare it with the alignment factor of previous data. The following Table 11 shows the Alignment Factors. When the deterioration in the alignment factor as mentioned in Sec 7.2 happens, then the threshold value Eq. (61) in the Sec 6.4 is set to a different value (here it has been set to 0.8) and the algorithm of Section 6 run again to see if the alignment factor is increased. If there is no increase, then we have to divide this

Online Data Sequence of data into small subsets. For the evaluation purposes, we consider only data sets 3 and 4 because of their huge size. These results are summarized in the following Table 8.11.

Table 8.11
Alignment Factors and Decision

Datasets	Size of Online Data Sequence 5	Alignment Factor till online data Sequence 4	Alignment Factor till online data Sequence 5
Dataset3	4 P, 142 N	1.14	1.32
Dataset4	2 P, 246 N	1.78	1.46

From the above Table 11, it is clear that there the alignment factor for the 5'th online batch of data has improved for the online dataset 3. So, the algorithm is effective in handling the online data in the case of dataset 3. However, the alignment factor degraded in the case of dataset4 when the 5'th online batch of data is trained. This is because; the total size of the data set 4 with 5'th batch of online data is 1305 which is a huge amount of data. We followed the algorithm presented in Sec. 7.3 to determine the optimum chunk of data to be considered at a time and this was determined to be 82 (3 batches of size 82 each = 246) in the case of Dataset 4. The classification accuracy for all the 3 batches of online dataset 5 is shown in the following figure 8.1.

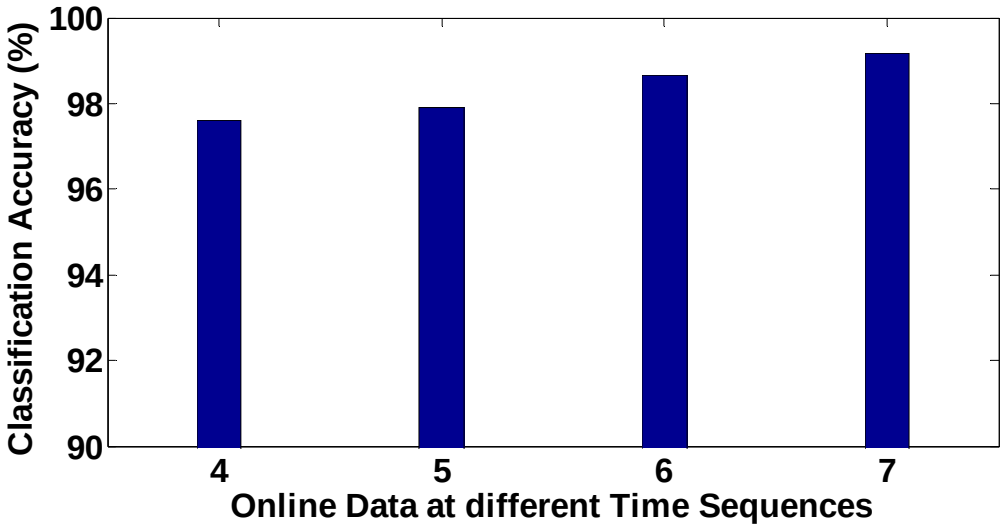

Figure 8.1 Classification Accuracy of Dataset4 for time sequences of 4 and above

This shows that the proposed method is efficient in handling very large online data over long time sequences. After finite number of sequences, the classification performance of the online data sets is approaching that of initial offline training batch of data. This indicates that training the subsequent online batches of data according to their nature is advantageous over statistical offline leanrning.

8.5. Computational Savings

Since we do not consider all the online data that becomes available after the initial online training, our proposed method yields huge savings in terms of computational time. Figure 8.2 shows the computational savings of our proposed method over considering all the online data without discarding it based on its nature. In the figure, "Data without considering the heterogeneous nature" indicates the traditional statistical learning and "Data with considering the heterogeneous nature" indicates our proposed method.

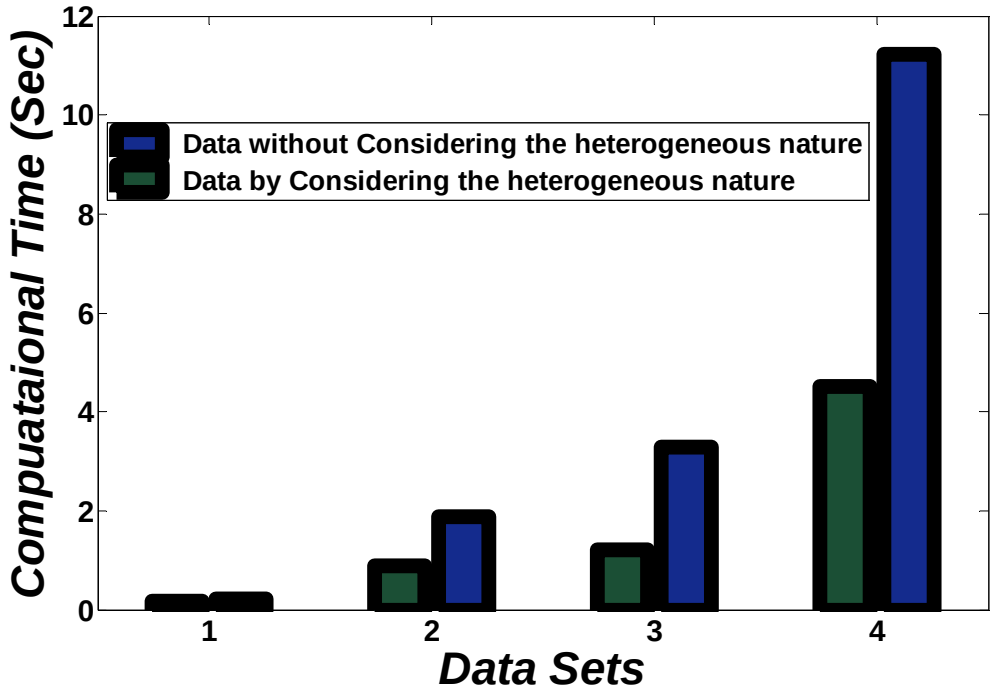

Figure 8.2 Data Sets Vs Computational Time (Sec)

This figure shows that for the larger data sets, the proposed dynamic learning algorithm along with AKFA resulted in Computational savings over 50% when it is used for training the new patterns of data when compared with the traditional offline or statistical learning. That is it takes us half the time to perform dynamic online learning as that of traditional offline learning. So this dynamic online learning is computationally very efficient.

CHAPTER 9 CONCLUSION and ACKNOWLEDGEMNT

9.1. Conclusion

Our thesis proposes a novel method of handling the long time heterogeneous trajectories of online data that uses the information of off-line training data or the first available batch of data (offline learning is done according to methods mentioned in Chapter 4). This method used along with Accelerated Kernel Feature Analysis (AKFA), a faster and more efficient feature extraction algorithm derived from the Kernel Principle Component Analysis (KPCA) is an efficient means for the detection of polyps on CT colonographic images. The polyp classification experiment using a *k*-nearest neighbor classifier showed that our proposed method along with AKFA gave comparable performance in discrimination negatives of online data (trained according to its nature) with that of the initial offline training data and also resulted in computational savings. Hence our model produced minimal false positives. However, since the large online data sets consisted of small number of true positives, the discrimination ability of true positives by AKFA was low and remained consistent with the initial kernel adaptation analysis mentioned in Chapters 4 and 5. This demonstrates that the features extracted by AKFA will be practically useful in discrimination of polyps from false positives detections for large online data when the data contains large number of positives. In that case our proposed model has the potential to lead a model-based CAD scheme yielding high detection performance of polyps and such a CAD scheme has the potential of making CT a viable option for screening large patient populations, resulting in early detection of colon cancers and leading to reduced mortality due to colon cancer.

9.2. Acknowledgment

This study was supported by *"Institutional Research Grant IRG-73-001-31 from the American Cancer Society."*

LITERATURE CITED

[1] S. Winawer, R. Fletcher, D. Rex, J. Bond, R. Burt, J. Ferrucci, T. Ganiats, T. Levin, S. Woolf, D. Johnson, L. Kirk, S. Litin, and C. Simmang, "Colorectal cancer screening and surveillance: clinical guidelines and rationale-Update based on new evidence," *Gastroenterology,* vol. 124, pp. 544-60, Feb 2003. Systems, vol. 36, no. 1, pp. 266 - 278, Jan 2000.

[2] K. D. Bodily, J. G. Fletcher, T. Engelby, M. Percival, J. A. Christensen, B. Young, A. J. Krych, D. C. Vander Kooi, D. Rodysill, J. L. Fidler, and C. D. Johnson, "Nonradiologists as second readers for intraluminal findings at CT colonography," Acad Radiol, vol. 12, pp. 67-73, Jan 2005

[3] J. G. Fletcher, F. Booya, C. D. Johnson, and D. Ahlquist, "CT colonography: unraveling the twists and turns," *Curr Opin Gastroenterol,* vol. 21, pp. 90-8, Jan 2005.

[4] H. Yoshida and A. H. Dachman , "CAD techniques, challenges and controversies in computed tomographic colonography", Abdom Imaging, vol. 30, pp. 26-41, Jan – Feb 2005.

[5] H. Yoshida and J. Näppi, "Three-dimensional computer-aided diagnosisscheme for detection of colonic polyps," *IEEE Trans Med Imaging*, vol. 20, pp. 1261-74, Dec 2001.

[6] R. M. Summers, C. F. Beaulieu, L. M. Pusanik, J. D. Malley, R. B. Jeffrey, Jr., D. I. Glazer, and S. Napel, "Automated polyp detector for CTcolonography: feasibility study," *Radiology*, vol. 216, pp. 284-90, 2000.

[7] R. M. Summers, M. Franaszek, M. T. Miller, P. J. Pickhardt, J. R. Choi, and W. R. Schindler, "Computer-aided detection of polyps on oral contrast-enhanced CT colonography," *AJR Am J Roentgenol*, vol. 184, pp. 105-8, Jan 2005.

[8] G. Kiss, J. Van Cleynenbreugel, M. Thomeer, P. Suetens, and G. Marchal, "Computer-aided diagnosis in virtual colonography via combination of surface normal and sphere fitting methods," *Eur Radiol*, vol. 12, pp. 77-81, Jan 2002.

[9] D. S. Paik, C. F. Beaulieu, G. D. Rubin, B. Acar, R. B. Jeffrey, Jr.,J. Yee, J. Dey, and S. Napel, "Surface normal overlap: a computer-aided detection algorithm with application to colonic polyps and lung nodules in helical CT," *IEEE Trans Med Imaging*, vol. 23, pp. 661-75, Jun 2004.

[10] A. K. Jerebko, R. M. Summers, J. D. Malley, M. Franaszek, and C. D.Johnson, "Computer-assisted detection of colonic polyps with CT colonography using neural networks and binary classification trees," *Med Phys*, vol. 30, pp. 52-60, Jan 2003.

[11] J. Näppi, H. Frimmel, A. H. Dachman, and H. Yoshida, "A newhigh-performance CAD scheme for the detection of polyps in CT colonography," *Medical Imaging 2004: Image Processing*, 2004, pp. 839-848.

[12] A. K. Jerebko, J. D. Malley, M. Franaszek, and R. M. Summers, "Multiple neural network classification scheme for detection of colonic polyps in CT colonography data sets," *Acad Radiol*, vol. 10, pp. 154-60, Feb 2003.

[13] A. K. Jerebko, J. D. Malley, M. Franaszek, and R. M. Summers,"Support vector machines committee classification method for computer-aided polyp detection in CT colonography," *Acad Radiol*, vol. 12, pp. 479-86, Apr 2005.

[14] V. N. Vapnik, The nature of statistical learning theory, 2nd ed New York: Springer, 2000.

[15] R. Courant and Hilbert, “Methods of Mathematical Physics,” vol 1, pp. 138-140,1966.

[16] O. L. Mangasarian, A.J. Smola, and B. Schölkopf, "Sparse kernel feature analysis," *University of Winconsin, Tech. rep. 99-04*, 1999.

[17] J. Franc and V. Hlavac, "Statistical Pattern Recognition Toolbox for Matlab,", *http://cmp.felk.cvut.cz/~xfrancv/stprtool/*, 2004.

[18] "Partners Research Computing," *http://www.partners.org/rescomputing/*, 2006.

[19] A. J. Smola, B. Schölkopf, "Sparse Greedy Matrix Approximation for Machine Learning", *Proc. 17th International Conf. on Machine Learning, 2000.*

[20] X. Jiang, Y. Motai, R. Snapp, and X. Zhu, Accelerated Kernel Feature Analysis, *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 109-116, 2006.

[21] B. Schölkopf and A. J. Smola, Learning with Kernels, MIT Press, 2002.

[22] Richard O. Duda, Peter E. Hart, David G. Stork.: Pattern Classification (2nd Edition), John Wiley & Sons Inc., 2001.

[23] Bernhard Schökopf , Alexander J. Smola.: Learning with Kernels:Support Vector Machines,Regularization, Optimization, and Beyond (Adaptive Computation and Machine Learning), MIT press, 2002

[24] H. Fröhlich, O. Chapelle, B. Scholkopf.: Feature selection for supportvector machines by means of genetic algorithm, Tools with Artificial Intelligence, Proceedings. 15th. IEEE International Conference, pp. 142 – 148, 2003

[25] Xue-wen Chen.: Gene selection for cancer classification using bootstrapped genetic algorithms and support vector machines, The Computational Systems, Bioinformatics Conference. Proceedings IEEE International Conference, pp. 504 – 505, 2003.

[26] Chanho Park and Sung-Bae Cho.: Genetic search for optimal ensemble of feature-classifier pairs in DNA gene expression profiles, Neural Networks, 2003. Proceedings of the International Joint Conference, vol.3, pp. 1702 – 1707, 2003.

[27] Firooz A. Sadjadi "Polarimetric Radar Target Classification Using Support Vector Machines" Optical engineering 47(4), 046201 April 2008.

[28] S. Amari and S. Wu,"Improving Support Vector Machine Classifiers by Modifying Kernel Functions", Neural Networks, Vol.6, pp.783-789, 1999

[29] H. Xiong, M.N.S. Swamy, and M.O. Ahmad, "Optimizing the Data- Dependent Kernel in the Empirical Feature Space," IEEE Trans. Neural Networks, vol. 16, pp. 460-474, 2005

[30] Huilin Xiong, Ya Zhang, and Xue-Wen Chen Data-Dependent KernelMachines for Microarray Data Classification. In Proceedings of IEEE/ACM Transactions on Computational Biology and Binformatics, vol. 4, NO. 4, October-December 2007.

[31] Tom Briggs, Tim Oates, "Discovering Domain Specific Composite kernels", Proceedings of the 20'th national conference on artificial Intelligence, p.732-738, July 09-13,2005, Pottsburgh, Pennsylvania

[32] N. Cristianini, J. Kandola, A. Elisseeff, and J. Shawe-Taylor, "On kernel target alignment," in *Proc. Neural Information Processing Sysytems(NIPS'01),* pp.367-373

[33] Sotiriou C, Wirapati P, Loi S, Harris A et al. Gene expression profiling in breast cancer: understanding the molecular basis of histologic grade to improve prognosis. J Natl Cancer Inst 2006 Feb 15; 98(4):262-72.PMID: 16478745

[34] M.A. Shipp, K.N. Ross, P. Tamayo, A.P. Weng, J.L. Kutok, R.C.T. Aguiar, M. Gaasenbeek, M.Angelo, M. Reich, G.S. Pinkus, T.S. Ray, M.A. Koval, K.W. Last, A. Norton, T.A. Lister, J. Mesirov, D.S. Neuberg, E.S. Lander, J.C. Aster, and T.R. Golub, "Diffuse Large B-Cell Lymphoma Outcome Prediction by Gene Expression Profiling and Supervised Machine Learning," Nature Medicine, vol. 8, pp. 68-74, 2002.

[35] Chandran UR, Ma C, Dhir R, Bisceglia M et al. Gene expression profiles of prostate cancer reveal involvement of multiple molecular pathways in the metastatic process. BMC Cancer 2007 Apr 12;7:64. PMID: 17430594

[36] Yu YP, Landsittel D, Jing L, Nelson J et al. Gene expression alterations in prostate cancer predicting tumor aggression and preceding development of malignancy. J Clin Oncol 2004 Jul 15;22(14):2790-9. PMID: 15254046

[37] B. J. Kim, I. K. Kim, K. B. Kim. Feature extraction and classificationsystem for nonlinear and online data, *Advances in Knowledge Discovery and Data mining, Proceedings*, volume 3056, pages: 171-180, 2004.

[38] W. Zheng, C. Zou, and L. Zhao. An improved algorithm for kernelprincipal component analysis. *Neural Process. Lett.*, 22(1):49–56, 2005.

[39] J. Kivinen, A. J. Smola, R. C. Williamson. Online learning with kernels, *IEEE Transactions on Signal Processing*, volume 52, Issue 8, pages 2165-2176, 2004.

[40] S. Ozawa, S. Pang, N. Kasabov. Incremental learning of chunk data for online pattern classification systems, *IEEE Transactions on Neural Networks*, volume 19, Issue 6, pages 1061-1074, 2008.

[41] H. T. Zhao, P. C. Yuen, J. T. Kwok. A novel incremental principal component analysis and its application for face recognition, *IEEE Transactions on Systems, Man, and Cybernetics* Part B-Cybernetics, volume 36, Issue 4, pages 873-886, 2006.

[42] Y. M. Li. On incremental and robust subspace learning, *Pattern Recognition*, volume 37, Issue 7, pages 1509-1518, 2004.

[43] Y. Kim. Incremental principal component analysis for image processing, *Optics Letters*, volume 32, Issue 1, pages 32-34, 2007.

[44] B. J. Kim, I. K. Kim. Incremental nonlinear PCA for classification,*Knowledge Discovery in Databases (PKDD 2004), Proceedings*, volume 3202, pages 291-300, 2004.

[45] B. J. Kim, J. Y. Shim, C. H. Hwang, I. K. Kim, J. H. Song. Incrementalfeature extraction based on empirical kernel map, *Foundations of Intelligent Systems*, volume 2871, pages 440-444, 2003.

[46] L. Hoegaerts, L. De Lathauwer, I. Goethals, J. A. K. Suykens, J.Vandewalle, B. De Moor. Efficiently updating and tracking the dominant kernel principal components, *Neural Networks*, volume 20, Issue 2, pages 220-229, 2007.

[47] T. J. Chin, D. Suter. Incremental kernel principal component analysis, *IEEE Transactions on Image Processing*, volume 16, Issue 6, pages 1662-1674, 2007.

[48] X. Jiang, Y. Motai, R.R. Snapp, and X. Zhu, Accelerated Kernel FeatureAnalysis, *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 109-116, 2006.

[49] Dense KNN software, "www.autonlab.org", 1993.

APPENDIX A

A.1 Acronyms Definitions

CAD	Computer Aided Detection
CTC	Computed Tomographic Colonography
KPCA	Kernel Principle Composite Analysis
AKFA	Accelerated Kernel Feature Analysis
RF	Residue Factor

A.2 Symbol Definitions

x_i	Input Data
y_i	Output Class label
$k_{ij} = \langle \Phi(x_i), \Phi(x_j) \rangle$	Element of Gram Matrix
K	Kernel Gram matrix
λ	Eigenvalue
$p_r(x_i, x_j)$	Input Data Kernel Matrix
$k_r(x_i, x_j)$	Data-Dependant Kernel
$Q(.)$	Factor Function
$k_0(x_i, x_j)$	Base Kernel Matrix
α	Combination Coefficient
S_{br}	Between – Class Scatter Matrices
S_{wr}	Within – Class Scatter Matrices
J	Class Separability
λ_*	Maximum Eigenvalue

$K_{comp}(\rho)$	Composite Kernel Gram Matrix
ρ	Composite Coefficient
$A(k, yy^T)$	Alignment Factor
$\langle K, yy^T \rangle_F$	Frobenius inner product between K and yy^T.
$J(\rho)$	Raleigh Coefficient for Composite Coefficient
$\hat{\rho}$	Composite coefficient vector that maximizes $J(\rho)$
x_i'	New Online Data
$p_r'(x_i, x_j)$	Input Kernel Matrix with new online data.
$k_r'(x_i, x_j)$	Data-Dependant Kernel Matrix with new online data.
$J'(\alpha')$	Class Separability with new online data
α'	Combination Coefficient with new online data
λ_*'	Maximum Eigenvalue with new Online data
m_1	Mean of class1 input data
m_2	Mean of class 2 input data
ξ	Parameter that determines the nature of the online data.
Rf	Residue Factor
$\overline{\alpha_*'}$	Mean of maximum Combination coefficient vector out of 4 kernels with online data
$\overline{\alpha_*}$	Mean of maximum Combination coefficient vector out of 4 kernels of off-line data.
Δ_{rk}	Difference Vector
$\min(\Delta_{rk})$	Minimum Difference Vector
$\alpha_r^{n'}$	Updated Combination Coefficient Vector
$Q_r^{n'}$	Updated Factor Function Matrix

$P_r^{n'}$	Updated Input Kernel gram Matrix
$K_r^{n'}$	Updated Data Dependant Kernel Gram Matrix
$K_{comp}^{n'}(\rho)$	Updated Composite Kernel Gram Matrix
$y^{n'}$	Updated output label matrix
$\hat{\rho}'$	New Coefficient Vector for creating Updated Composite Kernel
$J(\rho')$	Updated Raleigh Coefficient for Composite Kernel.
${}^{t}(P_r^{n'})_0$	Input matrix from time 0 to time *t* for all 4 kernels
${}^{t}(Q_r^{n'})_0$	Factor function matrix from time 0 to time *t* for all 4 kernels
${}^{t}(\alpha_r^{n'})_0$	Combination Coefficient from time 0 to time *t* for all 4 kernels
${}^{t+1}(P_r^{d1})_t$	Input matrix at time *t+1* which includes data from subset *d1*
${}^{(t+1)}(\alpha_r^{d1})_t$	Combination Coefficient Vector at time *t+1* which includes *d1* data
${}^{(t+1)}(J^{d1}(\alpha_r))_t$	Class Separability after the including *d1* data at time *t+1*
${}^{t}(\lambda_*)_0$	Maximum Eigenvalue of the data-dependant kernels from time 0 to time *t*
${}^{(t+1)}(\lambda_*')_t$	Maximum Eigenvalue of the data-dependant kernels from time *t* to time *t+1*
${}^{(t+1)}(\xi)_t$	Parameter that determines the nature of the online data at time *t+1*
${}^{t+1}(rf^{d1})_t$	Residue Factor with d1 subset of data from time at time *t+1*
$\overline{{}^{t}(\alpha_*)_0}$	Mean of Maximum Combination

	Coefficient of data Kernels till time *t*
$\overline{{}^{(t+1)}(\alpha_*^{d1})}_t$	Mean of Maximum Combination Coefficient of data Kernels from time *t* till time *t+1*
${}^{(t+1)}(\lambda_*^{d1})_t$	Maximum Eigenvalue of the data-dependant kernels of data *d1* from time *t* to time *t+1*
${}^{t+1}(\Delta_r^{d1})_t$	Difference Vectors of *d1* data from time *t* to *t+1* for all kernels.
${}^{t}n_0$	Size of data-dependant Kernel Matrices till time *t*
l	Size of *d1* data
${}^{t+1}(P_r^{n'})_t$	Updated Input Kernel Matrix at time *t+1*
${}^{t+1}(Q_r^{n'})_t$	Updated Factor Function Matrix at time *t+1*
${}^{(t+1)}(K_r^{n'})_t$	Updated Data-Dependant Kernel Matrices at time *t+1*
$\rho_i^{n'}$	Updated Composite Coefficient for composite Kernel Matrix at time *t+1*
${}^{t}(K_{comp}^{n'})_0$	Composite Kernel Matrix from time 0 till time *t*
${}^{t}(A'(k_{comp}^{n'}, y_{n'}, y_{n'}^T))_0$	Alignment Factor from time 0 till time *t*
${}^{t+1}(A'(k_{comp}^{n'}, y_{n'}, y_{n'}^T))_t$	Alignment Factor from time *t* to time *t+1*
${}^{t+1}(K_{comp}^{n'})_t$	Composite Kernel Matrix from time *t* to time *t+1*

APPENDIX B

B.1 Matlab code for Cross – Validation

```
function cross_classification()
clc; clear all;

TP_Train_Portion=stptrn; % stptrn is the proportion of the data that is
used as training set. For Eg: stptrn = 0.8
TP_Test_Portion=stptst; % stptst is the proportion of the data that is
used as training set. For Eg: stptrn = 0.2
FP_Train_Portion= sfptrn; %sfptrn is the proportion of the data that is
used as test set. For Eg: stptrn = 0.7
FP_Test_Portion= sfptst; % sfptst is the proportion of the data that is
used as test set. For Eg: stptrn = 0.3


%load ConvertedData
load newTP
[dim,TP_Total]=size(TP);

load newFP
[dim,FP_Total]=size(FP);
tempFP=zeros(dim,FP_Total);
tempFP=FP;
%size(TP)
%size(FP)


TP_train_size=round(TP_Total*TP_Train_Portion)
TP_test_size=round(TP_Total*TP_Test_Portion)
FP_train_size=round(FP_Total*FP_Train_Portion)
FP_test_size=round(FP_Total*FP_Test_Portion)

trndata.X=[tempFP(:,1:FP_train_size) TP(:,1:TP_train_size)];
trndata.size=TP_train_size+FP_train_size;
tstdata.X=[tempFP(:,(1+FP_train_size):(FP_train_size+FP_test_size))
TP(:,(1+TP_train_size):(TP_train_size+TP_test_size))];
tstdata.size=TP_test_size+FP_test_size;
tstdata.FPsize = FP_test_size;
tstdata.TPsize = TP_test_size;
trndata.FPsize = FP_train_size;
trndata.TPsize = TP_train_size;


% clear tempFP
```

```
trndata.X=(trndata.X-
ones(dim,1)*mean(trndata.X))./(ones(dim,1)*std(trndata.X));
% Normalization
trndata.y=[zeros(1,FP_train_size), ones(1,TP_train_size)];
tstdata.X=(tstdata.X-
ones(dim,1)*mean(tstdata.X))./(ones(dim,1)*std(tstdata.X));
tstdata.y=[zeros(1,FP_test_size), ones(1,TP_test_size)];

%defining the kernels and their arguments to be used in the
construction of gram matrices

%rbf kernel
options.ker1 = 'rbf'; % uses RBF kernel
options.arg1 = pr1; % pr1 = parameter of RBF kernel

%polynomial, Linear and Sigmoid kernels
options.ker2 = 'poly'; %uses Polynomial kernel.
%options.ker2 = 'linear'; % uses Linear Kernel. For the linear kernel,
the parameters are zero by default.
%options.ker2 = 'sigmoid'; % uses Sigmoid Kernel.
options.arg2 = pr2;%
options.arg3 = pr3;%

%k0 kernel
options.ker0 = 'rbf';
options.arg0 = pr0;

%definig dimensions
options.new_dim = d; % d = output dimension. Eg. d = 100.
[Accuracy,Confusion]=cross_validation(mno,options,trndata,tstdata,kno);
% mno represents the model number we decide to choose, kno (usually kno
= 10) represents the number of nearest neighbors to be used. mno takes
the values of 1 and 2 where 1 represents KPCA, 2 represents AKFA.
```

B.2 Matlab code for Cross – Classification

```
function
[accuracy,confusion]=cross_validation(method,options,trndata,tstdata,nu
m_neighbor)

accuracy=zeros(1,num_neighbor); % [1,20] here num_neighbor = kno
confusion=zeros(2,2);           % [2,2] indicates the dimensions of the
confusion matrix.

switch method
    case 1
        model=MyKPCA(trndata.X,options);  % KPCA
    case 2
        model=AKFA(trndata.X,options,0.0);  % AKFA
end

trnreduced=kernelproj(trndata.X,model);

for k=1:num_neighbor
```

```
  TPs(k) = 0;
  TNs(k) = 0;
  FNs(k) = 0;
  FPs(k) = 0;
  sensitivity(k) = 0;
  specificity(k) = 0;
  classification_accuracy(k) = 0;
end

wrong=zeros(num_neighbor,tstdata.size);
for i=1:tstdata.size
    tstreduced=kernelproj(tstdata.X(:,i),model);
    tmp1= sum(( trnreduced-(tstreduced*ones(1,(trndata.size))) ).^2,1);
    [y,id1]=sort(tmp1);

    for k=1:num_neighbor
        nnlabels=trndata.y(id1(1:k)); % Labels of nearest neighbors
        for j=1:2
            countlabels(j)=length(find(nnlabels==(j-1)));
        end
        [y,id2]=max(countlabels);  % Check the mostly selected cluster,
id2 can be 1 or 2

        if id2==(tstdata.y(i)+1)
            accuracy(k)=accuracy(k)+1;
            if i>tstdata.FPsize
                TPs(k) = TPs(k) + 1;
            else
                TNs(k) = TNs(k) + 1;
            end
        else
            wrong(k,i)=1;  % Incorrectly classified
            confusion(tstdata.y(i)+1,id2) =
confusion(tstdata.y(i)+1,id2) + 1;
            if i>tstdata.FPsize
                FNs(k) = FNs(k) + 1;
            else
                FPs(k) = FPs(k) + 1;
            end
        end   % end of (id2==(tstdata.y(i)+1))
        %if k==10
            %disp([i (id2-1) tstdata.y(i) TPs(k) TNs(k) FNs(k) FPs(k)])
      % end
     end  % end of (k=1:num_neighbor)
end  % end of (i=1:tstdata.size)

for k=1:num_neighbor
  sensitivity(k) = TPs(k)/(TPs(k) + FNs(k));
  specificity(k) = TNs(k)/(TNs(k) + FPs(k));
  classification_accuracy(k) = (TPs(k) + TNs(k))/(TPs(k) + TNs(k) +
FPs(k) + FNs(k));
end

nsensitivity = sensitivity;
nspecificity =specificity;
naccuracy = accuracy/tstdata.size
```

```
save('wrong.mat','wrong');
figure(1);
plot(naccuracy);
hold on
%plot(nsensitivity,'r-');
%plot(nspecificity,'b-');

legend('Accuracy'); % legend('Accuracy', 'Sensitivity', 'Specificity');
ylabel('Accuracy);
%title('Accuracy, Sensitivity and Specificity vs. Number of
Neighbors');
xlabel('Number of Neighbors'); % Plot against num_neighbor values
```

B.3 Matlab code for Kernel Projection

```
function out_data=kernelproj(in_data, model)
% KERNELPROJ Kernel projection.
% Synopsis:
%  Y = kernelproj(X, model)
%  out_data = kernelproj(in_data, model)

% Description:
%  Y = kernelproj(X, model) this function maps input vectors
%  X [dim x num_data] onto vectors Y [new_dim x num_data]

%    using the kernel projection

%  Y(:,i) = Alpha' * kernel(sv.X, X(:,i), ker, arg) + b

%    where parameters of the projection are given in model:

%     .Alpha [nsv x new_dim] Multipliers.
%     .b [new_dim x 1] Bias.
%     .sv.X [dim x nsv] Vectors.
%     .options.ker [string] Kernel identifier.
%     .options.arg [1 x narg] Kernel argument.

%  out_data = kernelproj(in_data, model) assumes that in_data is a
structure containing vectors X and labels y. The output structute
out_data is constructed as

%     out_data.X = kernelproj(in_data.X, model)
%     out_data.y = in_data.y

% Example:
%  help kpca;
%  help gda;
%
% See also
%  GDA, KPCA, LINPROJ, KERNEL.
```

```
%

% About: Statistical Pattern Recognition Toolbox
% (C) 1999-2003, Written by Vojtech Franc and Vaclav Hlavac
% <a href="http://www.cvut.cz">Czech Technical University Prague</a>
% <a href="http://www.feld.cvut.cz">Faculty of Electrical
Engineering</a>
% <a href="http://cmp.felk.cvut.cz">Center for Machine Perception</a>

% Modifications:
% 19-sep-2004, VF, core of the function rewritten to C
% 14-may-2004, VF
% 4-may-2004, VF

if isstruct(in_data)==1,
  out_data = in_data;

  if ~isempty(model.Alpha) & isfield(model, 'Alpha'),
    out_data.X = kernelproj_mex(in_data.X, model.Alpha, model.b, ...
        model.sv.X, model.options.ker, model.options.arg);
  else
    [dim,num_data]=size(in_data.X);
    out_data.X = model.b*ones(1,num_data);
  end

else
  if ~isempty(model.Alpha) & isfield(model,'Alpha'),

    out_data = kernelproj_mex(in_data, model.Alpha, model.b, ...
        model.sv.X, model.options.ker, model.options.arg);
  else
    [dim,num_data]=size(in_data);
    out_data = model.b*ones(1,num_data);
  end
end

return;
```

APPENDIX C

C.1 Matlab code for Data Dependent Composite Kernel

```
bk1 = kernel(X,options.ker1,options.arg1);  % X is the training data.
This creates the first Kernel Matrix that is built with the kernel
function specified in ‘ker1’ with parameters defined in arg1. In this
particular case it is RBF kernel.

%bk1 = (bk1-
ones(length(bk1),1)*mean(bk1))./(ones(length(bk1),1)*std(bk1)); This is
used for Normalization

bk2 = kernel(X,options.ker2,[options.arg2 options.arg3]); %In this case
the kernel function can be any one of Polynomial, Linear or Sigmoid
Kernel functions. Lalace Kernel is constructed as follows:

%bk2 = zeros(a1,a1); %where a is the size of the train data.
%a1 = TP_test_size+ FP_test_size;
%sigma = s1; %s1 is the parameter for the Laplace Kernel.
%for i = 1:a1
    %for j = 1:a1

        %inval=(X(i,i)-2*X(i,j)+X(j,j));
        %if inval<0;
            %ival = -1*inval;
        %else
             %ival = inval;
        %end
            %bk2(i,j) = exp(-sigma*(ival^0.5));

    %end
%end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
bk0 = kernel(X,options.ker0,options.arg0);% This is used for the
construction of K0 kernel matrix mentioned in eq. 28
d = ones(length(bk0),1);
k1 = [d,bk0];

% now creating mo,no, bo,wo matrices...for Kernel1;
%Let TP = TP_train_size and FP = FP_train_size;

A11 = 1/FP*bk1(1:FP,1:FP);
A22 = 1/TP *bk1(FP+1:FP+TP,FP+1:FP+TP);
a = zeros(FP,TP);
```

```
b = zeros(TP,FP);
B1 = [A11 a; b A22] - 1/a1*bk1;
W1 = diag(diag(bk1)) - [A11 a; b A22];

% now creating mo,no, bo,wo matrices...for Kernel2;

D11= 1/FP*bk2(1:FP,1:FP);
D22= 1/TP*bk2(FP+1:FP+TP,FP+1:FP+TP);
B2 = [D11 a; b D22] - 1/a1*bk2;
W2 = diag(diag(bk2)) - [D11 a; b D22];


M1 = k0'*B1*k0;
 M1 = (M1-ones(length(M1),1)*mean(M1))./(ones(length(M1),1)*std(M1));
 %Normalizing

N1 = k0'*W1*k0;%+0.1* eye(length(M1));
 N1 = (N1-ones(length(M1),1)*mean(N1))./(ones(length(M1),1)*std(N1));
  %Normalizing

M2 = k0'*B2*k0;
 M2 = (M2-ones(length(M1),1)*mean(M2))./(ones(length(M1),1)*std(M2));
  %Normalizing

N2 = k0'*W2*k0;%+0.1* eye(length(M2));
N2 = (N2-ones(length(M1),1)*mean(N2))./(ones(length(M1),1)*std(N2));
  %Normalizing


%Finding the eigen values to determine the dominant kernels.
Illustrating this process for two kernels.

[alp1,lambda1]= eig(M1,N1);
[c1,i1] = max(diag(lambda1));
AL1 = alp1(i1,:)'; %AL1 is the combination vector for the first kernel
function.
c1 %eigen value

[alp2,lambda2]= eig(M2,N2);
[c2,i2] = max(diag(lambda2));
AL2 = alp2(i2,:)'; %AL2 is the combination vector for the second kernel
function.

c2 %eigen value

c = max(c1,c2);

%Optimizing Combination coefficient vector so as to maximize the class
seperability.

for n = 1:c %c is any arbitrary number that denotes the number of
iterations.

    j11= AL1'*M1*AL1;
    j12= AL1'*N1*AL1;
```

```
    j1 = j11/j12;
    j21= AL2'*M2*AL2;
    j22= AL2'*N2*AL2;
    j2 = j21/j22;

%at present i am not using the decreasing rate for eta. But if chose to
do so, we may use
     % n1 = 0.01*(1-n/5);
      %n2 = 0.005*(1-n/5);

      AL1= AL1+ 0.01(1/j12*M1 - j1/j12*N1)*AL1;%for gaussian kernel;
 AL2= AL2 + 0.005(1/j22*M2 - j2/j22*N2)*AL2;%for all other kernels;


q1 = k0*AL1;
q2 = k0*AL2;

Q1= diag(q1);
Q2= diag(q2);
K1 = Q1*bk1*Q1;
K2=  Q2*bk2*Q2;


%%Now since we have the gram matrices K1 and K2, we can now proceed
towards linearly combining them.

%part b:

y=[zeros(1,FP) ones(1,TP)]';

%y denotes the class labels. In this particular example, all the false
polyps are represented by a '0' and all the true positives are
represented by a '1'.

y1= y*y';%The matrix yy' should be order nxn
u1 = trace(K1'*y1);
u2 = trace(K2'*y1);
U1 = [u1*u1 u1*u2; u2*u1 u2*u2];
V = [trace(K1'*K1) trace(K1'*K2); trace(K2'*K1) trace(K2'*K2)];

% If the matrix V is singular then we will use eig(U,V) to compute the
% eigen values and eigen vectors. Otherwise we simply use eig(inv(V)*U)
to compute the eigen vectors and values respectively because now the
generalized eigen problem is reduced to a standard eigen value problem

%Checking whether the matrix V is singular or not.
%1)If its rank is equal to its order then it is a non singular matrix.
%2) its determinant should be a non zero value.


order = max(size(V));

if rank(V)==order&& det(V)~=0
   [rho,lambda3]= eig(inv(V)*U1);
else
   [rho,lambda3]= eig(U1,V);
```

```
end

%rho = rho';
[c3,i3] = max(diag(lambda3));

reqrho = rho(i3,:);
s = exp(reqrho(1,1))+ exp(reqrho(1,2));

% if the coefficients of the composite coefficient vector are negative,
then we follow the following steps to produce correct result.

if reqrho(1,1)<0 && reqrho(1,2)<0
    reqrho1 = exp(reqrho)./s;
else if reqrho(1,1)>0 && reqrho(1,2)<0
    reqrho1 = [reqrho(1,1) exp(reqrho(1,2))/s];
    else if reqrho(1,1)<0 && reqrho(1,2)>0
              reqrho1 = [exp(reqrho(1,1))/s reqrho(1,2)];
        else reqrho1 = exp(reqrho)./s;
        end
    end
end

reqrho1
r1 =reqrho1(1,1);
r2 = reqrho1(1,2);
K = reqrho1*K1+ reqrho2*K2;%Linear Combination of the data dependant
gram matrices yielding a composite kernel;

%Linear Combination of the data dependant gram matrices yielding a
composite kernel;

% Once, the Gram matrix K is ready, we can use it with either KPCA or
AKFA in the sections D.1 and D.2 of Appendix D.
```

C.2 Matlab code for Online Kernel Adaptation Analysis

```
bk10 = kernel(x,options.ker1,options.arg1);
%x is the existing (or offlie) data + online data.

bk20 = kernel(x,options.ker2,[options.arg2 options.arg3]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%

bk00 = kernel(x,options.ker0,options.arg0);%
d = ones(length(bk00),1);
k0 = [d,bk3];

% now creating mo,no, bo,wo matrices...for Kernel1;
%Let TP0 = TP_train_size of the new data and the previous existing data
and FP0 = FP_train_size of the new data and the previous existing data;

A110 = 1/FP0*bk1(1:FP0,1:FP0);
```

```
A220 = 1/TP0*bk1(FP0+1:FP0+TP0,FP0+1:FP0+TP);
a = zeros(FP0,TP0);
b = zeros(TP0,FP0);
B10 = [A110 a; b A220] - 1/a10*bk10; %a10 is the size of the new train
data + previous offline data.
W10 = diag(diag(bk10)) - [A110 a; b A220];

% now creating mo,no, bo,wo matrices...for Kernel2;

D110= 1/FP0*bk20(1:FP0,1:FP0);
D220= 1/TP0*bk20(FP0+1:FP0+TP0,FP0+1:FP0+TP0);
B20 = [D110 a; b D220] - 1/a10*bk20;
W20 = diag(diag(bk20)) - [D110 a; b D220];


M10 = k00'*B10*k00;
 M10 = (M10-
ones(length(M10),1)*mean(M10))./(ones(length(M10),1)*std(M10));
 %Normalizing

N10 = k00'*W10*k00;%+0.1* eye(length(M1));
 N10 = (N10-
ones(length(M10),1)*mean(N10))./(ones(length(M10),1)*std(N10));
  %Normalizing

M20 = k00'*B20*k00;
 M20 = (M20-
ones(length(M10),1)*mean(M20))./(ones(length(M10),1)*std(M20));
  %Normalizing

N20 = k00'*W20*k00;%+0.1* eye(length(M2));
N20 = (N20-
ones(length(M10),1)*mean(N20))./(ones(length(M10),1)*std(N20));
  %Normalizing


%Finding the eigen values to determine the dominant kernels.
Illustrating this process for two kernels.

[alp10,lambda10]= eig(M10,N10);
[c10,i10] = max(diag(lambda10));
AL10 = alp10(i10,:)'; %AL1 is the combination vector for the first
kernel function.
c10% eigen value of the resultant train data

[alp20,lambda20]= eig(M20,N20);
[c20,i20] = max(diag(lambda20));
AL20 = alp2(i20,:)'; %AL2 is the combination vector for the second
kernel function.

c20 %eigen value of the resultant train data
c0 = max(c10,c20);

%rf = (mean(mean(AL10))-mean(mean(AL1)))*(c0/c);
```

```
if c0>c
   AL10 = AL1;
   bk10 = bk1;
   bk20 = bk2;
   K10 = K1;
   K20 = K2;

else if c0<c&&(mean(mean(AL10))-mean(mean(AL1)))*(c0/c)<1;
         %for the new online data that are fp.

         dfp1 = zeros(1,FPoff);%FPoff = size of Offline FP train data
         dfp2 = zeros(1, FPoff);

             for j = 1:FPon %size of FP's in online data
                  dfp1 = [AL10(1: FPoff)-AL10(FPoff +j)];
                  dfp2 = [AL20(1: FPoff)-AL20(FPoff +j)];
                  if dfp1 < 0
                        dfp1 = -1*dfp1;
                  else
                         dfp1 = dfp1;

                        if dfp2 < 0
                              dfp2 = -1*dfp2;
                        else
                              dfp2 = dfp2;

                        [d1 ind1]=min(dfp1);
                        [d2 ind2]=min(dfp2);

                        AL10(ind1)=AL10(FPoff +j);
                         AL20(ind2)=AL10(FPoff +j);

                        bk10 = [bk10(1:(ind1-1),:);(bk10((ind1+1):(a10
                        -j),:))];

                         bk10 = [bk10(:,1:(ind1-1)) (bk10(:,(ind1+1):(
                        a10 -j)))];

                         bk20 = [bk20(1:(ind2-1),:);(bk20((ind2+1):(
                        a10-j),:))];

                        bk20 = [bk20(:,1:(ind2-1)) (bk2(:,(ind2+1):(
                        a10 -j)))];

                         end
                  end
            end



         AL10FP = [AL10(1:FPoff,:)];
         AL20FP = [AL20(1:FPoff,:)];
         size(AL10FP);
```

```
%for the new online data that are fp.

dtp1 = zeros(1,TPoff);
dtp2 = zeros(1,TPoff);

for j = 1:TPon
   dtp1 = [AL10((FP0ff+FPon)+1:a10)-AL10((a10-TPon)+j)];
   dtp2 = [AL2(FP0ff+FPon)+1): a10)-AL20((a10-TPon)+j)];
   if dtp1 < 0
         dtp1 = -1*dtp1;
   else
         dtp1 = dtp1;

         if dtp2 < 0
               dtp2 = -1*dtp2;
         else
               dtp2 = dtp2;

               [d1 ind1]=min(dtp1);
               ind1 = ind1+69;
               AL10(ind1)=AL10((a10-TPon)+j);

               [d2 ind2]=min(dtp2);
               ind2 = ind2+(FPoff+FPon);
               AL20(ind2)=AL20((a10-TPon)+j);

               bk10 = [bk10(1:(ind1-
               1),:);(bk10((ind1+1):((a10-FPon)-j),:))];

               bk10 = [bk10(:,1:(ind1-1))
               (bk10(:,(ind1+1):((a10-FPon)-j)))];

               bk20 = [bk20(1:(ind2-
               1),:);(bk20((ind2+1):((a10-FPon)-j),:))];

               bk20 = [bk20(:,1:(ind2-1))
               (bk20(:,(ind2+1):((a10-FPon)-j)))];

               bk00 = [bk00(1:(ind1-
               1),:);(bk00((ind1+1):((a10-FPon)-j),:))];

               bk00 = [bk00(:,1:(ind1-1))
               bk00(:,(ind1+1):((a10-FPon)-j)))];


        end
    end
end

tpbk1 = size(bk1)
AL10TP = [AL10((FPoff+FPon+1):(a10-TPon))];
AL20TP = [AL20((FPoff+FPon+1):(a10-TPon))];
size(AL10TP);
AL10 = [AL10FP' AL10TP']';
AL20 = [AL20FP' AL20TP']';
```

```
size(AL10);

d = ones(length(bk00),1);
k0 = [d,bk3];


A110 = 1/FPoff*bk10(1:FPoff,1:FPoff);
A220 = 1/TPoff*bk10(FPoff+1:a1, FPoff+1:a1);
a = zeros(FPoff,TPoff);
b = zeros(TPoff,FPoff);
B10 = [A110 a; b A220] - 1/a1*bk10;
W10 = diag(diag(bk10)) - [A110 a; b A220];

D110= 1/FPoff*bk20(1:FPoff,1:FPoff);
D220= 1/TPoff*bk20(FPoff+1:a1, FPoff+1:a1);
B20 = [D110 a; b D220] - 1/a1*bk20;
W20 = diag(diag(bk20)) - [D110 a; b D220];

M10 = k00'*B10*k00;
M10 = (M10-
ones(length(M10),1)*mean(M10))./(ones(length(M10),1)*std(M10)
);

N10 = k00'*W10*k00;%+0.1* eye(length(M1));
      N10 = (N10-
  ones(length(M10),1)*mean(N10))./(ones(length(M10),1)*std(N1
  0));

M2 = k00'*B20*k001;
M20 = (M20-
ones(length(M10),1)*mean(M20))./(ones(length(M10),1)*std(M20)
);

N20 = k10'*W20*k10;%+0.1* eye(length(M2));
N20 = (N20-
ones(length(M10),1)*mean(N20))./(ones(length(M10),1)*std(N20)
);


[alp1,lambda1]= eig(M10,N10);
[c10,i10] = max(diag(lambda1));
AL10 = alp1(i10,:)';

[alp2,lambda2]= eig(M20,N20);
[c20,i20] = max(diag(lambda2));
AL20 = alp2(i20,:)';

c10
c20

j110= AL10'*M10*AL10;
j120= AL10'*N10*AL10;
j10 = j110/j120;
j210= AL20'*M20*AL20;
j220= AL20'*N20*AL20;
```

```
          j20 = j210/j220;

         AL10= AL10+ (1/j120*M10 - j10/j120*N10)*AL10; %for Gaussian
        kernel;
         AL20= AL20 + (1/j220*M20 - j20/j220*N20)*AL20;

         q10 = k10*AL10;
         q20 = k20*AL20;

         Q10= diag(q10);
         Q20= diag(q20);
         K10 =  Q10*bk10*Q10;%/5;
         K20 = (Q20*bk20*Q20);%/8;


else
        j110= AL10'*M10*AL10;
        j120= AL10'*N10*AL10;
        j10 = j110/j120;
        j210= AL20'*M20*AL20;
        j220= AL20'*N20*AL20;
        j20 = j210/j220;

        AL10= AL10+ (1/j120*M10 - j10/j120*N10)*AL10; %for Gaussian
        kernel;
        AL20= AL20 + (1/j220*M20 - j20/j220*N20)*AL20;

        q10 = k10*AL10;
        q20 = k20*AL20;

        Q10= diag(q10);
        Q20= diag(q20);
        K10 = Q10*bk10*Q10;
        K20 = Q20*bk20*Q20;

end

% finding the composite coefficients of updated Gram matrices.

K1 = K10;
K2 = K20;

if c0>c
    FPtrainsize = FPoff;
    TPtrainsize = TPoff;

else if c0<c&&(mean(mean(AL10))-mean(mean(AL1)))*(c0/c)<1;

    FPtrainsize = FPoff;
    TPtrainsize = TPoff;

else
    FPtrainsize = FPoff+FPon;
    TPtrainsize = TPoff+TPon;
```

```
y=[zeros(1,FPtrainsize) ones(1,TPtraoinsize)]';

%y denotes the class labels. In this particular example, all the false
polyps are represented by a '0' and all the true positives are
represented by a '1'.

y1= y*y';%The matrix yy' should be order nxn
u1 = trace(K1'*y1);
u2 = trace(K2'*y1);
U1 = [u1*u1 u1*u2; u2*u1 u2*u2];
V = [trace(K1'*K1) trace(K1'*K2); trace(K2'*K1) trace(K2'*K2)];

% If the matrix V is singular then we will use eig(U,V) to compute the
% eigen values and eigen vectors. Otherwise we simply use eig(inv(V)*U)
to compute the eigen vectors and values respectively because now the
generalized eigen problem is reduced to a standard eigen value problem

%Checking whether the matrix V is singular or not.
%1)If its rank is equal to its order then it is a non singular matrix.
%2) its determinant should be a non zero value.


order = max(size(V));

if rank(V)==order&& det(V)~=0
   [rho,lambda3]= eig(inv(V)*U1);
else
   [rho,lambda3]= eig(U1,V);
end

%rho = rho';
[c3,i3] = max(diag(lambda3));

reqrho = rho(i3,:);
s = exp(reqrho(1,1))+ exp(reqrho(1,2));

% if the coefficients of the composite coefficient vector are negative,
then we follow the following steps to produce correct result.

if reqrho(1,1)<0 && reqrho(1,2)<0
    reqrho1 = exp(reqrho)./s;
else if reqrho(1,1)>0 && reqrho(1,2)<0
    reqrho1 = [reqrho(1,1) exp(reqrho(1,2))/s];
    else if reqrho(1,1)<0 && reqrho(1,2)>0
              reqrho1 = [exp(reqrho(1,1))/s reqrho(1,2)];
        else reqrho1 = exp(reqrho)./s;
        end
    end
end

reqrho1
r1 =reqrho1(1,1);
r2 = reqrho1(1,2);

K = reqrho1*K1+ reqrho2*K2;
```

```
%After we compute the updated Data Dependant Composite Kernel Gram
Matrix for the online data, we now use this K with AKFA described in
Appendix D.2.
```

APPENDIX D

D.1 Matlab code for KPCA

```
function model = MyKPCA(X,options)
% KPCA Kernel Principal Component Analysis.
%
% Synopsis:
%  model = kpca(X)
%  model = kpca(X,options)
%
% Description:
%  This function is implementation of Kernel Principal Component
Analysis (KPCA) [Schol98b]. The input data X are non-linearly mapped to
a new high dimensional space induced by prescribed kernel function. The
PCA is applied on the non-linearly mapped data. The result is a model
describing non-linear data projection. See 'help kernelproj' for info
how to project data.

% Input:
%  X [dim x num_data] Training data.

%  options [struct] Decribes kernel and output dimension:
%   .ker [string] Kernel identifier (see 'help kernel');
%     (default 'linear').
%   .arg [1 x narg] kernel argument; (default 1).
%   .new_dim [1x1] Output dimension (number of used principal
%     components); (default dim).

% Output:
%  model [struct] Kernel projection:
%   .Alpha [num_data x new_dim] Multipliers.
%   .b [new_dim x 1] Bias.
%   .sv.X [dim x num_data] Training vectors.
%
%   .nsv [1x1] Number of training data.
%   .eigval [1 x num_data] Eigenvalues of centered kernel matrix.
%   .mse [1x1] Mean square representation error of maped data.
%   .MsErr [dim x 1] MSE with respect to used basis vectors;
%      mse=MsErr(new_dim).
%   .kercnt [1x1] Number of used kernel evaluations.
%   .options [struct] Copy of used options.
%   .cputime [1x1] CPU time used for training.
%
% Example:
%  X = gencircledata([1;1],5,250,1);
%  model = kpca( X, struct('ker','rbf','arg',4,'new_dim',2));
%  XR = kpcarec( X, model );
%  figure;
%  ppatterns( X ); ppatterns( XR, '+r' );
```

```
% Modifications:
% 4-may-2004, VF
% 10-july-2003, VF, computation of kercnt added
% 22-jan-2003, VF
% 11-july-2002, VF, mistake "Jt=zeros(N,L)/N" repared
%              (reported by SH_Srinivasan@Satyam.com).
% 5-July-2001, V.Franc, comments changed
% 20-dec-2000, V.Franc, algorithm was implemented


% timer
start_time = cputime;

% gets dimensions
[dim,num_data] = size(X);

% process input arguments
%---------------------------------
if nargin < 2, options = []; else options=c2s(options); end
if ~isfield(options,'ker'), options.ker = 'linear'; end
if ~isfield(options,'arg'), options.arg = 1; end
if ~isfield(options,'new_dim'), options.new_dim = dim; end

% compute kernel matrix
 K = kernel(X,options.ker,options.arg);   % Kernels with only one
argument

%K = kernel(X,options.ker,[options.arg1 options.arg2]);   % Kernels
with two arguments

% Centering kernel matrix (non-linearly mapped data).
J = ones(num_data,num_data)/num_data;
Kc = K - J*K - K*J + J*K*J;

% eigen decomposition of the kernel matrix
[U,D] = eig(Kc);
Lambda=real(diag(D));

% normalization of eigenvectors to be orthonormal
for k = 1:num_data,
  if Lambda(k) ~= 0,
     U(:,k)=U(:,k)/sqrt(Lambda(k));
  end
end

% Sort the eigenvalues and the eigenvectors in descending order.
[Lambda,ordered]=sort(-Lambda);
ordered
save('indicesKPCA.mat','ordered');
Lambda=-Lambda;
U=U(:,ordered);
save('pcinorder.mat','U');

% use first new_dim principal components
A=U(:,1:options.new_dim);
```

```
save('firstpcs.mat','A');

% compute Alpha and compute bias (implicit centering)of kernel
projection
model.Alpha = (eye(num_data,num_data)-J)*A;
Jt=ones(num_data,1)/num_data;
model.b = A'*(J'*K*Jt-K*Jt);

% fill output structure
model.sv.X = X;
model.nsv = num_data;
model.options = options;
model.eigval = Lambda;
model.kercnt = num_data*(num_data+1)/2;
model.MsErr = triu(ones(num_data,num_data),1)*model.eigval/num_data;
model.mse = model.MsErr(options.new_dim);
model.cputime = cputime - start_time;
model.fun = 'kernelproj';

cputime_KPCA = model.cputime
% cputime-start_time
mse_KPCA = model.mse
return;
```

D.2 Matlab code for AKFA

```
function model=AKFA(X,options,delta)

num_data = size(X,2);
I=options.new_dim;
C=zeros(I,I);

start_time = cputime;  % for calculating the computational time

K=kernel(X,options.ker,options.arg); % Gram Matrix
J = ones(num_data,num_data)/num_data;
K = K - J*K - K*J + J*K*J; % Centering matrix

tempK=K; % tempK is the updated Gram Matrix

idx=zeros(1,I);
mark=ones(1,num_data); % =0 if i-th data selected as feature or deleted

for i=1:I
   if i==1
      c=sum(tempK.^2)./diag(tempK)';
   else
      c = zeros(1,num_data);
      for j=1:num_data
         if mark(j)>0
            if tempK(j,j)>delta
```

```
            c(j)=sum((tempK(j,:).*mark).^2)/tempK(j,j);
         else
            mark(j)=0;
         end
      end
   end
   end

   [CMax,idx(i)]=max(c);
   mark(idx(i))=0;
   for j=1:num_data % update the Gram Matrix
      if mark(j)>0
         tempK(j,:)=tempK(j,:)-
tempK(j,idx(i))*tempK(idx(i),:).*mark/(tempK(idx(i),idx(i)));
     end
   end

   C(i,i) = 1 / sqrt(tempK(idx(i),idx(i)));
   C(i,1:(i-1))=-sum(K(idx(i),idx(1:(i-1)))*C(1:(i-1),1:(i-1))'*C(1:(i-
1),1:(i-1)),1)*C(i,i);
end

cputime_AKFA =cputime-start_time

% fill output structure

save('indicesAKFA.mat','idx');

Alpha=zeros(num_data,I);
for i=1:I
   for j=1:I
      Alpha(idx(j),i)=C(i,j);
   end
end
model.Alpha=(eye(num_data,num_data)-J)*Alpha;

Jt=ones(num_data,1)/num_data;
model.b = Alpha'*(J*K*Jt-K*Jt);

model.sv.X = X;
model.options = options;
model.nsv = num_data;
model.coefficient=C;
model.cputime = cputime_AKFA;

Error=zeros(1,num_data);
for i=1:num_data
    Error(i)=K(i,i)-K(i,idx(1:I))*C'*C*K(idx(1:I),i);
end
Error;
model.mse=sum(Error)/num_data;
mse_AKFA = sum(Error)/num_data

% End of AKFA
```

APPENDIX E

E.1 Matlab code for ROC curves

```
function roctests()

% Xk = load('db2.xls','ascii'); % linear classifier on DB2
% roc_db2_125 =myroc(Xk(:,1:125),700,66,0,[-100:0.1:100],10000000,'linear');
% roc_db2_100 = myroc(Xk(:,1:100),700,66,0,[-00:0.1:100],10000000,'linear');
% roc_db2_050 = myroc(Xk(:, 1:50),700,66,0,   [-78:2:38],10000000,'linear');

Xk = xlsread('db3.xls');
% linear classifier on DB3
% roc_db3_058 = myroc(Xk(:,1:58),990,22,0,[-300  -2:2:58],1000000,'linear');
% roc_db3_050 = myroc(Xk(:,1:50),990,22,0,[-300  -2:2:82],1000000,'linear');
%roc_db3_030 = myroc(Xk(:,1:30),990,22,0,[-300 -50:2:100],1000000,'linear');
% plotroc(roc_db3_058,roc_db3_050,roc_db3_030,'dim=58','dim=50','dim=30');


% linear classifier on extended DB3

%roc_db3_058_ext = myroc(Xk(:,1:58),990,22,22,[-300 -42:2:70
100],1000000,'linear');
% roc_db3_050_ext = myroc(Xk(:,1:50),990,22,22,[-300   -72:2:90
100],1000000,'linear');
% roc_db3_030_ext = myroc(Xk(:,1:30),990,22,22,[-300 -220:2:120
150],1000000,'linear');
%plotroc(roc_db3_058_ext,roc_db3_050_ext,roc_db3_030_ext,'dim=58','dim=50','d
im=30');

% Elliptic classifier on extended DB3. The extended elliptic classifier for
DB3 is similar to that of Extended linear classifier.

roc_db3_058 = myroc(Xk(:,1:58),990,22,0,[],1000000,'elliptic');
roc_db3_050 = myroc(Xk(:,1:50),990,22,0,[],1000000,'elliptic');
roc_db3_030 = myroc(Xk(:,1:30),990,22,0,[],1000000,'elliptic');
plotroc(roc_db3_058,roc_db3_050,roc_db3_030,'dim=58','dim=50','dim=30');

function plotroc(r1,r2,r3,l1,l2,l3)
figure; hold on; grid on; axis([-0.1 1.1 -0.1 1.1]);
plot( (r1.FP)./(r1.FP+r1.TN), (r1.TP)./(r1.TP+r1.FN),'-r');
plot( (r2.FP)./(r2.FP+r2.TN), (r2.TP)./(r2.TP+r2.FN),'-g');
plot( (r3.FP)./(r3.FP+r3.TN), (r3.TP)./(r3.TP+r3.FN),'-b');
legend(l1,l2,l3,'Location','Best');

function stats = myroc(Xk,NN,NP,Next,b_values,tmax,classifier,extend)
data.X = Xk';
data.y = [ones(1,NN) 2*ones(1,NP)];
```

```
  posmodel = mlcgmm(data.X(:,(NN+1):(NN+NP)));
  randmat = rand(size(posmodel.Cov));
  posmodel.Cov = posmodel.Cov + 1e-8*randmat'*randmat; % needs to remain
positive semidefinite

  if Next>0 % extend positives training set
      data.X = [data.X gsamp(posmodel,Next)];
      data.y = [data.y 2*ones(1,Next)];
  end

  if strcmp(classifier, 'linear')
      fname = sprintf('lmodel_%i_%i_%03i.mat',NN,NP+Next,size(Xk,2));
      if exist(fname)
          load(fname,'lmodel');
      else
          lmodel = perceptron(data, struct('tmax',tmax));
          save(fname,'lmodel');
      end
      dfce = lmodel.W'*data.X;
  else
      tmp = data.X - repmat(posmodel.Mean,1,size(data.X,2));
      dfce = diag(tmp' * inv(posmodel.Cov) * tmp);
      b_values = -[0:0.001:1]*max(dfce(:));
  end

  for i=1:length(b_values) %for computing the confusion matrix i.e., no of
TP,TN,FP,FN
      y = ones(1,NN+NP+Next);
      y(find(dfce + b_values(i) < 0)) = 2;

      stats.TN(i) = length(find((data.y==1)& (y==1)));
      stats.TP(i) = length(find((data.y==2)& (y==2)));
      stats.FP(i) = length(find((data.y==1)& (y==2)));
      stats.FN(i) = length(find((data.y==2)& (y==1)));
      fprintf('%7.2f %4i %4i %4i
%4i\n',b_values(i),stats.TP(i),stats.TN(i),stats.FP(i),stats.FN(i));
  end
```